 Copy ______

IDA PAPER P-3166

TRANSLATING SYSTEM DEVELOPMENT DOCUMENTS TO HYPERTEXT: A GENERIC PROCESS AND LESSONS LEARNED FROM A PROTOTYPE

David A. Wheeler

Dennis W. Fife, *Task Leader*

December 1995

Prepared for
Ballistic Missile Defense Organization

IDA
INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Group Reports

Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

Papers

Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

Documents

IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract DASW01 94 C 0054 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

IDA PAPER P-3166

TRANSLATING SYSTEM DEVELOPMENT DOCUMENTS TO HYPERTEXT: A GENERIC PROCESS AND LESSONS LEARNED FROM A PROTOTYPE

David A. Wheeler

Dennis W. Fife, *Task Leader*

December 1995

IDA

INSTITUTE FOR DEFENSE ANALYSES

Contract DASW01 94 C 0054
Task T-R2-597.2

PREFACE

This document was prepared by the Institute for Defense Analyses (IDA) under the task order, Ballistic Missile Defense (BMD) Battle Management, Command, Control, and Communication (BM/C3) Planning and Assessments, and fulfills an objective of the task, to "finalize IDA's hypertext version of the NMD Information Architectures and document lessons learned to transfer the approach to similar BMC3 engineering information services."

This document was reviewed by the following IDA research staff members: Mr. John M. Boone, Dr. Alfred E. Brenner, Dr. Brian S. Cohen, Dr. Richard J. Ivanetich, Mr. Steve A. Lawyer, and Mr. Jonathan D. Wood. Mr. Matthew Hettermann, a summer student at IDA from the California Institute of Technology, helped to develop the prototype described in this document.

Table of Contents

List of Figures

EXECUTIVE SUMMARY

Purpose

The purpose of this paper is to assist the Ballistic Missile Defense Organization (BMDO) and others to use hypertext technology effectively. This paper presents a generic process and lessons learned for developing hypertext versions of system development documents. Such documents may be requirements documents, architectural documents, design documents, or other similar documents, and may have been developed using word processors, computer-aided software engineering (CASE) tools, drawing tools, or some combination of tools.

These documents describe many complex subcomponent interrelationships, making the information difficult to understand. Simply converting the contents of a document so it can be viewed by a hypertext tool does not necessarily improve the situation. Instead, what is required is a *tightly woven* document, that is, a hypertext document with a large number of internal hypertext interconnections that help the reader examine subcomponents and their interrelationships on line.

The generic process for developing tightly woven documents and the lessons learned described in this paper are based on our experiences in developing a prototype tightly woven document for BMDO.

Background

BMDO developed a document containing system development information, the *National Missile Defense (NMD) Battle Management, Command, Control and Communications (BM/C3) Information Architecture (IA)* [BMDO 1995]. The Institute for Defense Analyses (IDA) reviewed this document and determined that the document was difficult to understand and evaluate because it included a large number of interrelationships. These interrelationships were difficult to follow ("navigate"), and as a result the document was difficult to understand. Such difficulties are not unique to this document; many system development documents describe complex subcomponent interrelationships.

IDA recommended that BMDO investigate using hypertext technologies, such as the World Wide Web (the Web), to make it easier to understand system development documents. Hypertext and Web technologies have recently become widely available and at a low cost. BMDO agreed and asked IDA to demonstrate this approach.

Generic Process

This paper defines a generic process for developing tightly woven documents based on our prototyping effort. This process can be summarized in the following steps:

a. Determine desired results, available input, and external constraints.

b. Identify and select off-the-shelf (OTS) translation tools.

c. Identify and select techniques to add interconnections not added by OTS translation tools.

d. Implement programs and procedures to generate the desired results.

e. Test.

f. Iterate.

Lessons Learned

A number of lessons were learned on how to use hypertext technology, including the following:

a. Where possible, plan for generating a tightly woven document before developing the document.

b. Require consistent formatting of the original information because this simplifies the process of creating a tightly woven document.

c. Plan for human-generated information to have errors and occasional inconsistencies. Use tools to detect errors.

d. Design for information updates to be processed automatically where practical.

e. Tightly woven documents are practical and useful. Tightly woven documents make it easier to navigate the complex subcomponent interrelationships often present in system development documents.

A number of lessons were learned about the technology itself. Current technology is sufficient for the task, but there are limitations and difficulties that will probably be removed within the next few years. For example:

a. Current Web document formats are limiting.

b. Graphics are slow to display in some circumstances due to slow hardware/software systems, network delay, and insufficiently large caches.

Conclusions

Based on the prototyping effort, the user feedback from this effort, and the generic process and lessons learned derived from this effort, we conclude:

a. Complex documents with many internal interrelationships, such as many system design documents, can be made easier to navigate using hypertext to create tightly woven documents.

b. A generic process may be used for generating tightly woven documents.

c. The generic process is applicable to existing documents that were not intended to be used as hypertext documents. However, if the original documents were intended to be used as tightly woven documents, this process should be easier to perform.

d. New tools, such as HTML browsers, are now available that can significantly reduce the cost of developing and distributing system development information as tightly woven documents. There are limitations with some of these technologies, but they are sufficient to the task and many of these limitations will probably be eliminated over the next few years.

Recommendation

We recommend that BMDO consider, at document conception, developing and distributing key system development documents as tightly woven hypertext documents using the information discussed in this paper.

1. INTRODUCTION

1.1 PURPOSE

The purpose of this paper is to assist the Ballistic Missile Defense Organization (BMDO) in effectively using hypertext technology to navigate system development information. Other organizations interested in using hypertext may also find this paper useful.

This paper presents a generic process and lessons learned for developing hypertext versions of system development documents. Such documents may be requirements documents, architectural documents, design documents, or other similar documents, and may have been developed using word processors, computer-aided software engineering (CASE) tools, drawing tools, or some combination of tools.

A key issue is that these documents describe many complex subcomponent interrelationships, making the information difficult to understand. Simply converting its contents so it can be viewed by a hypertext tool does not necessarily improve the situation. Instead, what may be desired is a *tightly woven* document, that is, a hypertext document with a large number of internal hypertext interconnections that help the reader examine subcomponents and their interrelationships.

The generic process for developing tightly woven documents and the lessons learned described in this paper are based on our experiences in developing a prototype tightly woven document for BMDO.

1.2 BACKGROUND

Four factors set the stage for developing a prototype tightly woven document:

a. BMDO developed a document containing system development information, the *National Missile Defense (NMD) Battle Management, Command, Control and Communications (BM/C3) Information Architecture (IA)* [BMDO 1995].

b. Hypertext technologies have been maturing.

c. The World Wide Web (WWW) has accelerated the availability of hypertext.

d. A review of BMDO's IA by the Institute for Defense Analyses (IDA) recommended development of a hypertext version of the IA to make navigating system interrelationships easier [Fife 1995]. We call this hypertext prototype the *Hypertext IA*.

Each of these factors is discussed in the following paragraphs.

1.2.1 BMDO NMD BM/C3 Information Architecture

BMDO is pursuing an NMD program that "Develops and maintains the option to deploy a cost-effective, operationally effective, and Anti-Ballistic Missile Treaty compliant system at a single site as the initial step toward deployment of a system designed to protect the United States against limited Ballistic missile threats, including accidental or unauthorized launches or Third World attacks" [US 1995]. In support of this program there are a number of NMD BM/C3 activities, one of which has been the development of the IA.

The IA identifies the essential elements of command and control related information needed to accomplish a given mission. It specifies the information flow within the system, and the information needed from or provided to external sources. The intent of the IA is to support an incremental process to develop the necessary functionality with continual feedback from the user to ensure that the system ultimately delivered will meet the user's requirements and will perform as described in the user's concept of operations (CONOPS). The IA was developed using a variation of the Object Modeling Technique notation [Rumbaugh 1991]. This notation, like most notations, uses a combination of interconnected diagrams and supporting text to describe a system.

The NMD BM/C3 IA was developed in response to recommendations by the Defense Science Board, the Army Science Board, and the Air Force Scientific Advisory Board [DSB 1993, ASB 1995, AFSAB 1994]. They concluded that, coupled with the use of open architectures, best commercial practices, and standard off-the-shelf products, information architectures are important to developing BM/C3 systems capable of meeting evolving warfighter requirements on today's (and future) information-intensive battlefields.[1]

---

[1] The NMD BM/C3 IA has neither been reviewed nor approved by the Defense Science Board, the Army Science Board, nor the Air Force Scientific Advisory Board.

1.2.2 Hypertext

A hypertext document is a computerized collection of information consisting of many interconnected pieces of information. Each piece of information is called a "page" or a "node," and the interconnections between pages are called "links." An important aspect of good hypertext documents is that they need not be read sequentially (from beginning to end); instead, users can easily skip directly between the different portions that interest them.

Some people use the term "hypertext" for documents which only contain text and the term "hypermedia" to refer to documents that also contain graphics, sound, and/or video. In this paper, the term "hypertext" is used throughout and is considered synonymous with the term "hypermedia."

The history of hypertext begins with Vannevar Bush, science advisor to President Roosevelt during World War II. Bush proposed a hypertext system called Memex for making scientific information widely available [Bush 1945]. The term *hypertext* itself was coined in 1965 by Ted Nelson as part of his pioneering work on the "Xanadu" project. The goal of Xanadu was to distribute all of the world's written information electronically, interconnected as hypertext links [Nelson 1981]. Hypertext first became widely available in 1986 when Apple Corporation introduced its HyperCard product and included it free with every Macintosh sold. In November 1987 the Association for Computing Machinery held its first conference on hypertext [Nielson 1990].

1.2.3 World Wide Web (WWW)

The WWW, often called simply the "Web," is a hypertext system implemented on top of the Internet, and can be viewed as a combination of hypertext and transparent networking. Web technology (i.e. Web programs, protocols, and standards) provides the ability to access information as a large, world-wide collection of interlinked hypertext documents. For users, the most important type of program is a Web browser which retrieves hypertext pages for the user and provides the user interface. The most important Web data standard is Hypertext Markup Language (HTML), which defines the required data format for hypertext documents. A program that can view HTML files is called an HTML browser. Web browsers are a special kind of HTML browser that can also retrieve documents from an internet.

The Web originated at CERN as a mechanism to allow geographically distributed research physicists to more easily coordinate development of their scientific results. In

1989 Berners-Lee and Cailliau at CERN wrote the first Web project proposal.[2] By 1991 initial prototypes were released to a limited number of people. In 1993 the National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign, Illinois, released beta versions of the Mosaic browser for several common computer platforms as public domain software. Mosaic suddenly made the Web easy to use, visually interesting, inexpensive, and widely available.[3]

After this point the growth of the Web has been quite phenomenal. Figure 1 shows the growth of the Web, in terms of gigabytes transferred on the Web per month.[4]

Web technology can be used without making information (such as system development documentation) publicly available. Hypertext documents can be developed using Web data standards, yet only made available through private networks, disks, and tapes. Encryption, host access restrictions, and password protection can also be used to restrict access to information. This enables the use of widely available, low-cost tools while still protecting the information. The issue of security for system development documents is beyond the scope of this paper.

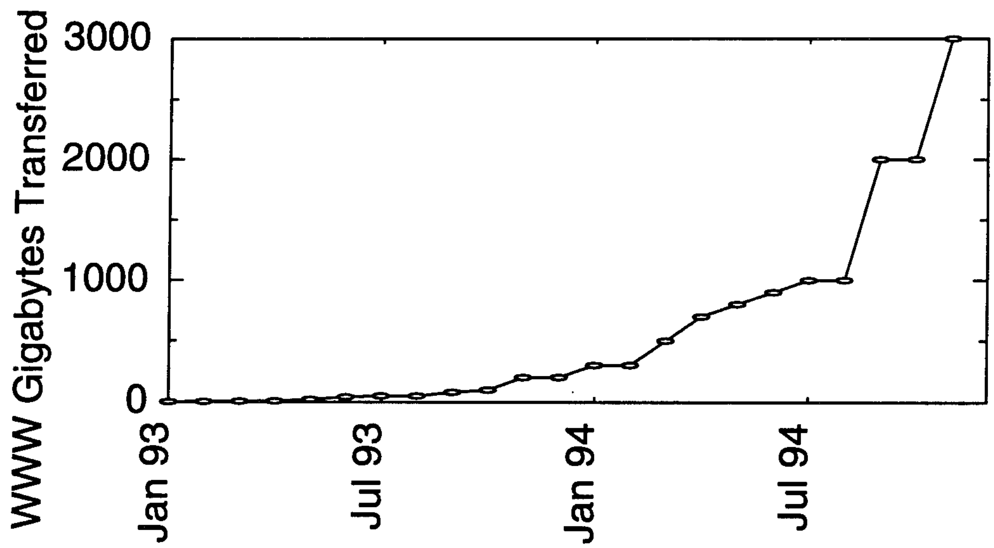

Figure 1. Growth of the Web

---

[2] CERN is the European Laboratory for Particle Physics; the acronym CERN comes from its older French name, Conseil Européen pour la Recherche Nucléaire, URL `http://www.cern.ch/`

[3] "A Little History," the World Wide Web Consortium (W3C), URL `http://www.w3.org/pub/WWW/History.html`

[4] This growth was estimated by the Internet Society, URL `http://www.isoc.org/`

1.2.4 IDA's Assessment of the IA

In 1995 IDA staff reviewed an early draft of a paper system development document, the NMD BM/C3 IA. The review found that the IA was difficult to read and understand because it contained a large number of interrelated components. Following these interrelationships, a process called "navigating," was time consuming and difficult in the paper document, and as a result the IA's quality was difficult to judge [Fife 1995]. Based on this review and other findings, IDA determined the following:

Methods for navigating system interrelationships should be improved. Many system development documents, not just the IA, are difficult to understand due to the many interrelationships between the system components. Improved methods for navigating complex interrelations would aid many system development tasks. BMDO's NMD program is emphasizing the use of executable representations, not just documents, but we believe that executable representations are unlikely to completely eliminate the need for system development documentation.

CASE tools alone do not solve the problem. Using CASE tools to view the system development documents can make it easier to navigate some system interrelationships, but such tools often cost tens of thousands of dollars per copy. Therefore, requiring every BM/C3 engineer and user to use a CASE tool is economically impractical for information with a large number of potential users. In addition, CASE tools often contain or reference a great deal of information stored as simple text. This text usually does not include connections to other system components. As a result, even with expensive CASE tools many important interrelationships may be difficult to navigate. CASE tools can be combined with other programs to create tightly woven hypertext documents, as is already occurring in some work by the BMDO System Engineering and Integration Contractor using ObjectMaker's ObjecTime CASE tool. Such work is covered by the generic process described in this paper.

Hypertext should be investigated. IDA recommended that BMDO investigate using hypertext capabilities to enable many concerned engineers and users to navigate complex system development documents such as the IA [Fife 1995]. Based on this recommendation, BMDO tasked IDA to develop a prototype to demonstrate these capabilities. At the time of task initiation Web technology was considered as an implementation vehicle because it was simple, inexpensive, and widely available.

1.3 APPROACH

Our approach was to learn by doing, with an emphasis on tightly woven results, to achieve our product and project goals.

1.3.1 Learning By Doing

Our approach to investigating how to develop hypertext versions of system development documents, such as the IA, was to "learn by doing." IDA developed a prototype Hypertext IA. A machine-readable copy of the IA was provided to IDA by the IA developers. IDA did not change the content of the IA itself but transformed it into a hypertext document by partitioning the information, creating numerous links to tie the information together, and reorganizing it to take best advantage of hypertext.

1.3.2 Emphasis on Tightly Woven Results

Tightly woven documents are hypertext documents with a large number of internal hypertext interconnections. In situations where there are many complex interconnections between actual system components, a correspondingly large number of hypertext links should be available to enable navigation to whatever information is important to the user and thus help the user understand the system.

Tightly woven documents are different from many system development documents currently available in hypertext formats. Many organizations use hypertext systems to distribute documents which are then downloaded, printed, and read sequentially. Many other organizations have hypertext documents that contain hypertext links, but most of the hypertext links are of only two kinds: (1) links based the document's table of contents, such as "next section" and "previous section," or (2) links to items outside the document. These uses of hypertext are quite acceptable in many circumstances, but our objective for using hypertext was different: to help users follow complex interconnections between components within a single system.

There is no conflict between these different uses of hypertext. It is possible to have a general BMDO BMC/3 "electronic library" with links and search capabilities to enable users to find the system development documents that best meet their needs, and then enable users to navigate those documents using tightly woven connections to learn whatever information they needed.

The term "tightly woven" can be misunderstood. Here are some important points to remember:

a. Any hypertext data format may be used to develop a tightly woven document. Such formats include HTML and those of some word processors.

b. A tightly woven document is a kind of hypertext document, but not all hypertext documents are tightly woven.

c. Tight weaving is a matter of degree above a minimum, not an absolute measure. A document with only four or less trivial links on each hypertext page (say to the next, previous, and "higher" document) would not be considered a tightly woven document. A document with only links based on its simple structure (i.e., a table of contents) would not be considered tightly woven. If hypertext links beyond these minimums are increasingly added to a hypertext document, the result can be considered increasingly tightly woven.

d. Tight weaving may not be appropriate for some documents or for some portions of documents. Hypertext links for their own sake are a waste of time—a large number of internal hypertext links should only be available if they add value for the document's users. There must be a determination that users will need to quickly follow interconnections in the way a tightly woven document permits, and that adding such interconnections is worth the time spent to create them.

e. Parts of a tightly woven document, or even an entire tightly woven document, can be created manually. A hypertext link can be created manually by identifying a specific link source and connecting it to a specific link destination. A tightly woven document can be created manually by repetitively creating hypertext links manually. A key criterion for choosing to do something manually should be its lifecycle cost: if it is cheaper over a document's lifecycle to add certain links manually, then they should be added manually. Creating a tightly woven document manually will become increasingly time consuming as the number of such links increases.

f. Word processors and CASE tools that generate hypertext documents do not necessarily eliminate the need for special techniques to create tightly woven documents. Word processors that support hypertext make it easy to create manual links, but they generally do not support automatic generation of such links. CASE tools can generate tightly woven documents from the data they control, but often the details of the system information are stored as ordinary text files. As a result, for these details CASE tools do not usually add the tight interconnections desired. In addition, most current CASE tools do not generate tightly woven documents, though this situation will probably improve. Therefore, approaches such as the one described in this paper still apply to information developed using word processors that support hypertext and CASE tools.

g. A tightly woven document is more navigable, but in some cases it may not be more understandable. Navigability can aid understanding, but understanding also depends on other factors such as the quality of the hypertext page contents.

h. The concept of "tightly woven" documents is in some sense not new. To some, this concept is obvious and is simply a description of a "good hypertext document." However, the concept is not universally applied where it would be appropriate to do so.

1.3.3 Goals of Prototype

The overall goal of the prototype Hypertext IA is to demonstrate that tightly woven documents are practical and useful. Its more specific goals are to:

a. Develop a tightly woven version of the NMD BM/C3 IA.

b. Handle document updates quickly and at low cost (due to changes in the NMD BM/C3 IA). This implies that the generation process must be highly automated since manual regeneration of an entire document after each revision would be too costly.

c. Require a small per-user cost. CASE tools are typically quite expensive ($10,000 or more per user), significantly reducing the number of potential users. A per-user cost of $50 or less for any necessary tools permits more users to access the information, as is the situation for most Web browsers.

d. Demonstrate the utility of tightly woven documents for system developers to navigate complex system development documents.

e. Aid in identifying techniques, processes, and lessons learned for developing tightly woven documents.

1.3.4 Goals of Project

The goals for this project are to successfully develop a prototype and then document the process sufficiently so others can repeat the process.

The project did not attempt to develop a cost model for the process of developing tightly woven documents. Many nonrecurring costs were involved, in particular identifying the overall process but also evaluating, installing, and learning how to use tools. Additionally, we had resources to develop only one document, and with only one data point we felt it was inappropriate to generate a cost model.

1.4 ORGANIZATION OF THIS DOCUMENT

Section 2 of this document describes the Hypertext IA as it appears to a user, providing an example of a tightly woven system development document. Section 3 describes the generic process used to implement the Hypertext IA, including examples. Section 4 describes the lessons learned from developing the Hypertext IA; these lessons learned should apply to other efforts at developing tightly woven documents. The lessons learned include lessons on how to use the technology as well as current limitations of hypertext technology. Section 5 gives an brief assessment of the Hypertext IA, showing that this prototype met its goals. This suggests that other documents developed using the same process may also meet these goals. Section 6 provides a set of conclusions and recommendations. Appendix A presents a sample script used to create the Hypertext IA. A list of references and a list of acronyms are provided at the end.

2. DESCRIPTION OF THE HYPERTEXT IA

This chapter describes the Hypertext IA's general organization, diagrams, and class descriptions, and depicts the ease and method in which users can navigate among many different types of related information in a tightly woven document.[5]

The paper form of the BM/C3 NMD IA, by contrast, is more difficult to use than the Hypertext IA. Every time a user wants to read a related piece of information in the paper version (e.g., to read the description of a class mentioned in a diagram), he or she must locate another page. While locating a single page is not terribly time consuming, following all related references is quite time consuming. Paper indexes can help but only to a limited degree: paper indexes require users to locate two pages for each reference. The Hypertext IA makes exploring related information easy: a single mouse click can quickly retrieve a related piece of information.

An HTML browser (e.g., Netscape or Mosaic) must be used to view the Hypertext IA. The browser shown in the following figures is Netscape Corporation's Netscape which shows hypertext links as underlined text or an outlined graphic image.

2.1 GENERAL ORGANIZATION

The first page the user sees is the Hypertext IA home page. A portion of this home page is shown in Figure 2. The Hypertext IA home page briefly describes the purpose of the prototype, references related information (such as the Defense Information Systems Agency's Global Command and Control System), and prominently lists a few carefully selected Hypertext IA pages. The prominent list allows users to jump directly to key sections of the IA document. These sections are:

a. The Executive Overview. This page describes the IA, why it was created, and the terminology used in the rest of the document. The executive overview is

[5] The Hypertext IA is available from IDA as a disk or tape that can be used on a local area network. The Hypertext IA is also available as part of BMDO's System Engineering Design Data Base (SEDD), a repository for BMDO system information to be remotely accessible through a BMDO-specific internet.

prominently listed because it provides novice IA users with a high-level summary of the information necessary to understand the rest of the IA.

b. The Table of Contents. The IA table of contents is similar to the paper document's table of contents. The table of contents shows the basic organization of the entire Hypertext IA and contains a large number of hypertext links to other portions of the Hypertext IA. This page is prominently listed because it provides a simple way to go to any part of the Hypertext IA. The table of contents is also helpful in keeping users from becoming "lost" in the details.

c. Introductions to the Object Views (Diagrams), the Lists of Diagrams, and the Dictionary of Classes. The primary technical content of the IA is the set of diagrams, diagram descriptions, and textual "class definitions" describing each diagram component in greater detail. These pages are prominently listed because experienced IA users will go straight to these items to quickly find specific pieces of information

After reading the executive overview, the user can go to the table of contents. The Hypertext IA table of contents, like a book's table of contents, shows the overall structure of the document and makes it easier to find specific items. All the items in this table provide links that connect directly to more specific information. Figure 3 shows a portion of the Hypertext IA table of contents.

Each of the prominently listed pages is easily accessible from any other location in the Hypertext IA. Every page in the Hypertext IA has a hypertext link back to the table of contents, and from the table of contents a user can select any of these prominently listed pages as well as other background pieces of information. All Hypertext IA pages have links to "next" and "previous" pages where appropriate, which makes it easier to access related information. Figure 4 shows the basic organization of the Hypertext IA, including some of the key hypertext links between top-level hypertext pages.

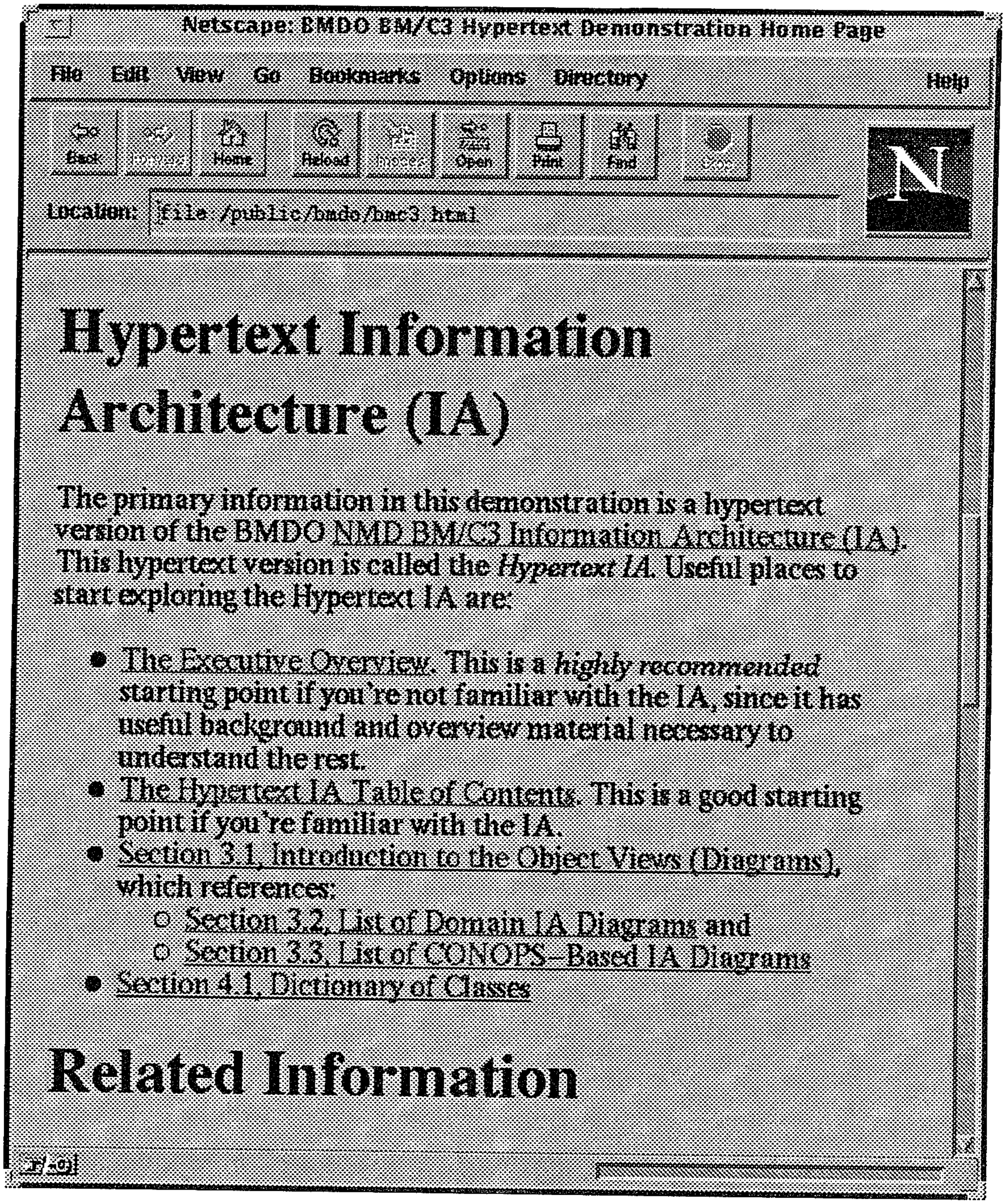

Figure 2. Screen Image: Hypertext IA Home Page

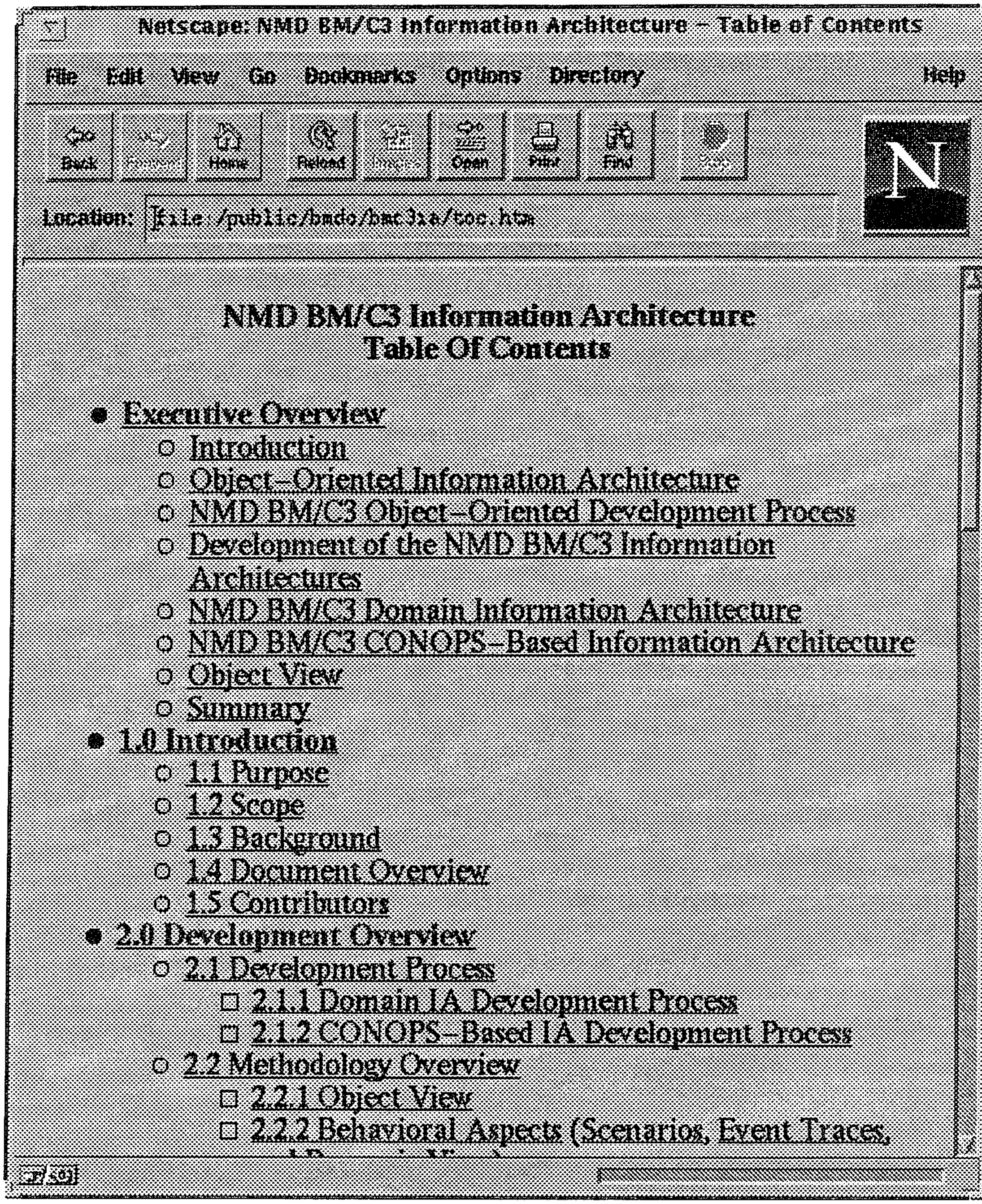

Figure 3. Screen Image: Hypertext IA Table of Contents

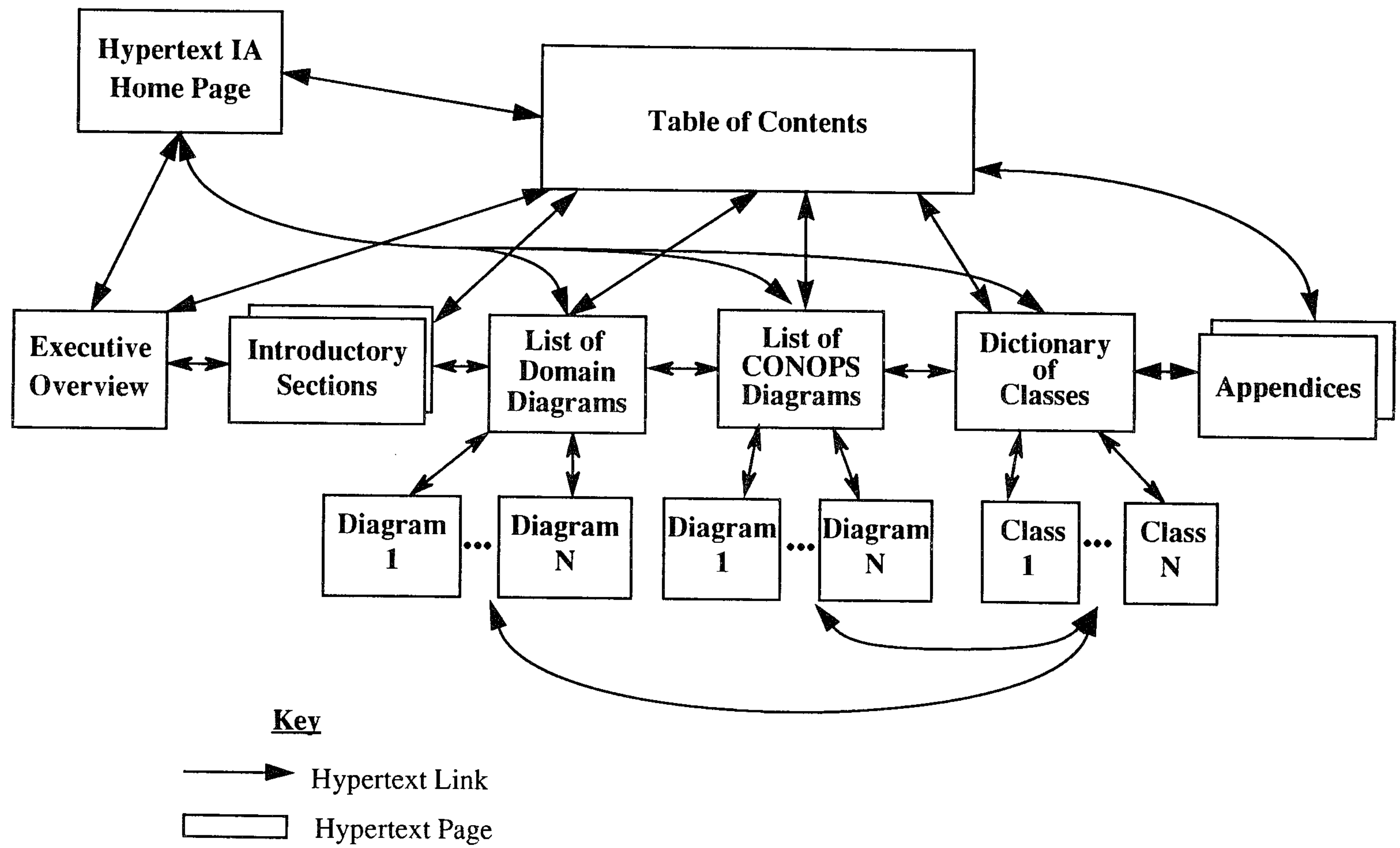

Figure 4. Hypertext IA Basic Organization

2.2 DIAGRAMS

Every diagram in the Hypertext IA can be accessed directly from the table of contents.[6] Each diagram shows a set of relationships (represented as labelled arcs) between a set of classes (represented as rectangles) using the Object Modeling Technique [Rumbaugh 1991]. Each diagram has supporting text describing the diagram. The IA actually contained two sets of diagrams: the "Domain" diagrams, which were diagrams to represent the classes required in any BM/C3 system, and the "CONOPS" diagrams, which were derived from the domain diagrams but expanded to include NMD-specific requirements from a concept of operations.

The diagrams are a good example of the kinds of restructuring necessary to produce a high quality result. One limitation of current Web browsers is that loading and displaying a large graphical diagram is a time-consuming operation. Rather than forcing users to wait for a diagram if they only wanted to read the diagram supporting text, we split each diagram into two hypertext pages, one containing the graphic and the other containing the supporting text. The supporting text has a hypertext link to its corresponding diagram and vice versa. Both the supporting text and diagrams have hypertext links to the "next" and "previous" diagrams as well as back to the table of contents.

Another limitation is that there is no single ideal graphical format. Ideally, a graphical format should support whatever screen resolution is available, support zooming in and out, support "embedding" a graphic in a larger hypertext page, and be viewable by most users without additional tools. No single graphical format currently meets all of these requirements. The solution we reached was to provide two formats for all important graphics: Adobe's Postscript and the Graphics Interchange Format (GIF). Postscript supports zooming in and out and is independent of the screen or printer resolution. GIF files can be embedded in hypertext pages and GIF viewers are built into most Web browsers.

Figure 5 illustrates how the diagrams supporting text and graphics (both formats) are organized. Figure 6 is a more detailed version of Figure 5, showing example text from the Hypertext IA, with the underlined text and dashed lines representing hypertext links.

Many of the diagrams are so complex that even high-resolution screens cannot legibly display an entire diagram at once. Since browsers (such as Netscape) provide scrollbars to view portions of a larger picture, this is not a significant problem. It should be noted,

---

[6] A user viewing image depicted in Figure 3 could use the scrollbars or page-down key to see the list of diagrams.

however, that the paper versions of these diagrams are unreadable or harder to use for the this reason.

Figure 7 shows a portion of a sample diagram. This diagram shows the classes Higher Authority, Rules of Engagement, User, Action, Action Information, and some of their interrelationships. Scrollbars at the right and bottom edges allow the rest of the diagram to be viewed. At the bottom of this diagram there are links to related pages, as shown in Figure 8.

Ideally, a user should be able to "click" on any specific item in a diagram to learn more about it. While such links are possible, it would be very difficult to create these links automatically and it would have been very time consuming to create all of these links "by hand" for this prototype. Instead, selecting any part of the diagram brings the user to the diagram's supporting text which contains links to every class referenced in the diagram. The end result is the same: users can quickly move from a diagram to more detail.

The original diagram descriptions did not list every class referenced in a given diagram, so IDA developed programs to add a list of classes. IDA also developed programs to create hypertext links from every class reference to its corresponding class description, as well as other relevant hypertext links.

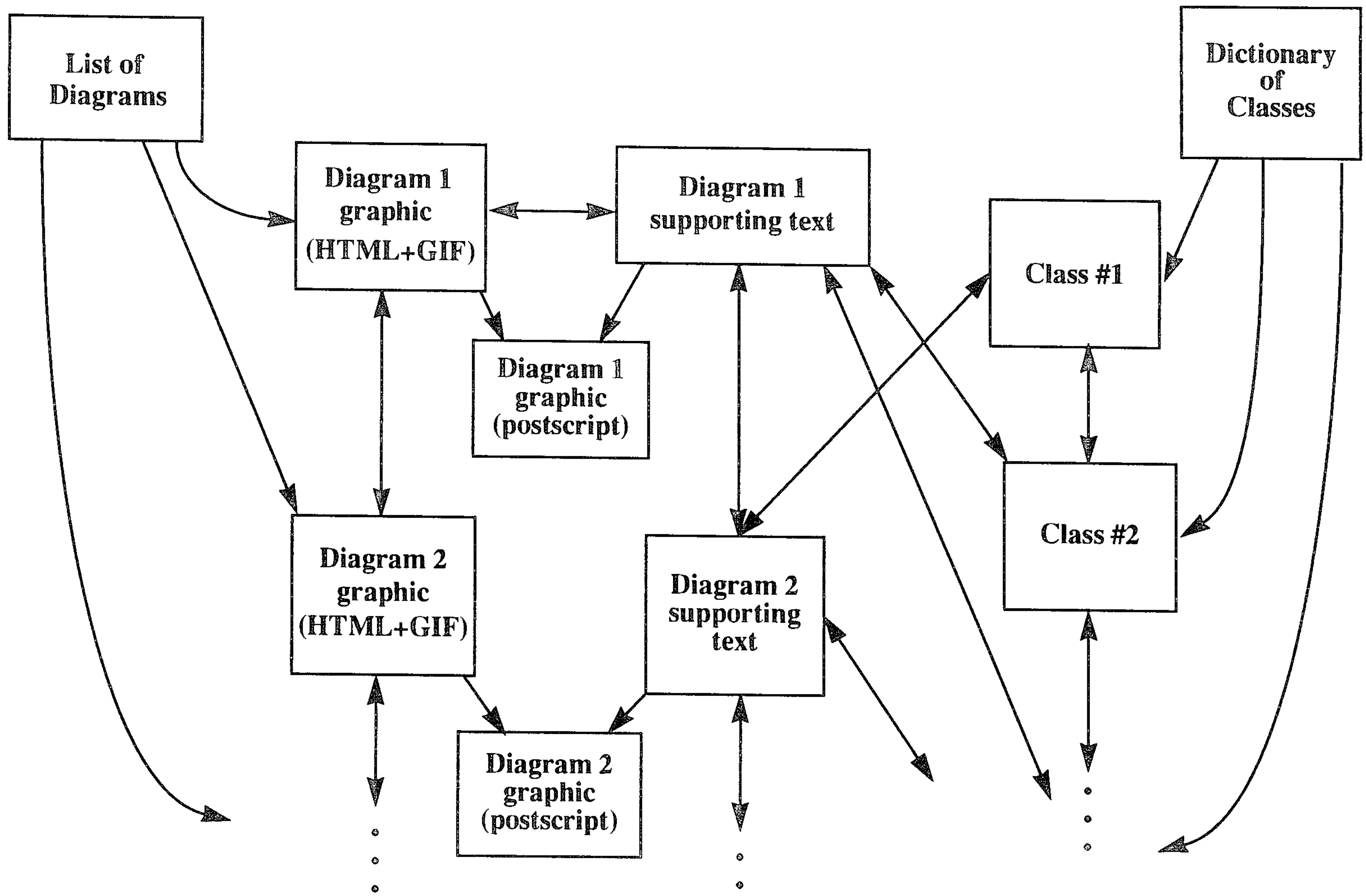

Figure 5. Connections of Diagrams and Class Descriptions

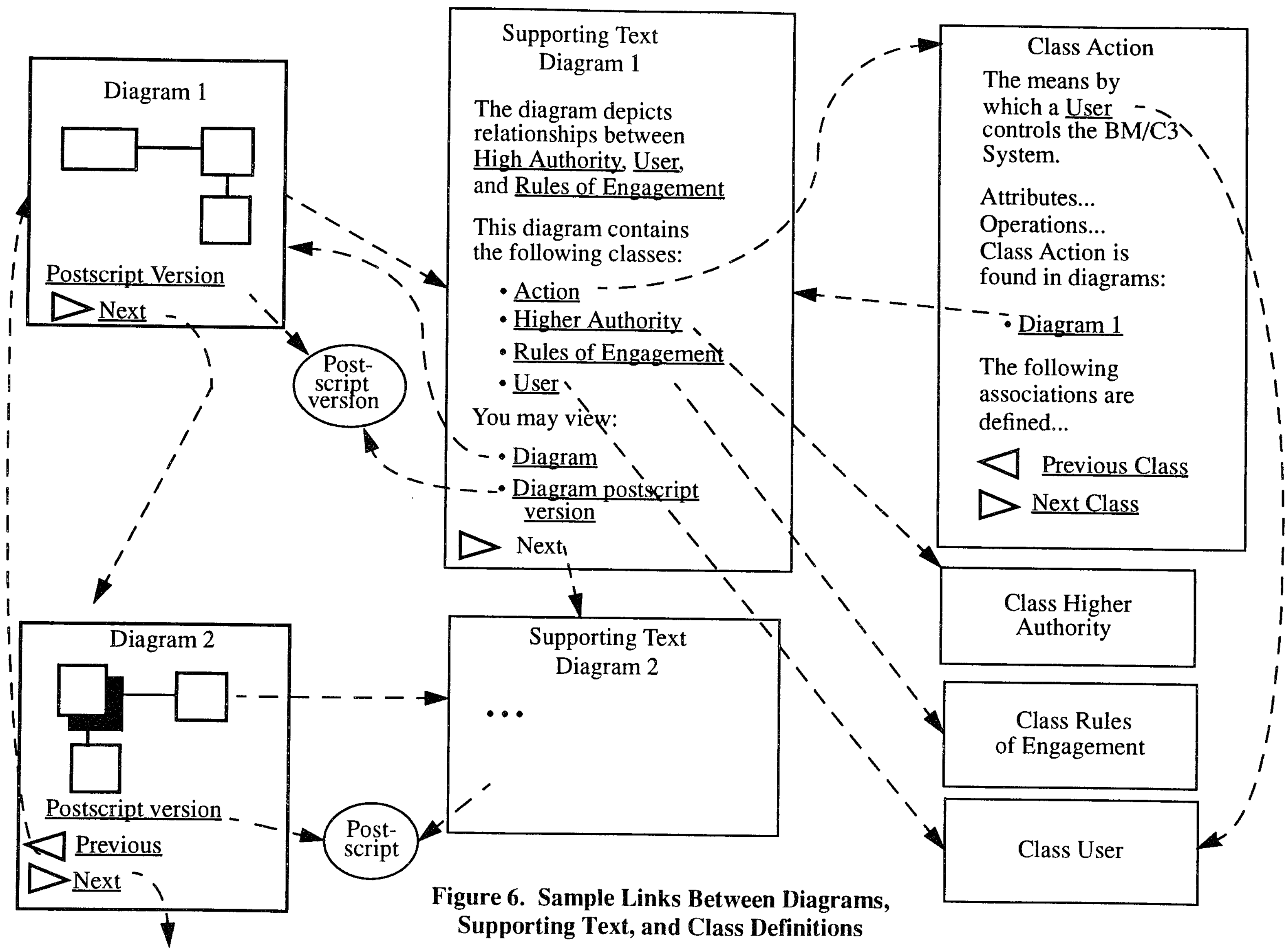

Figure 6. Sample Links Between Diagrams, Supporting Text, and Class Definitions

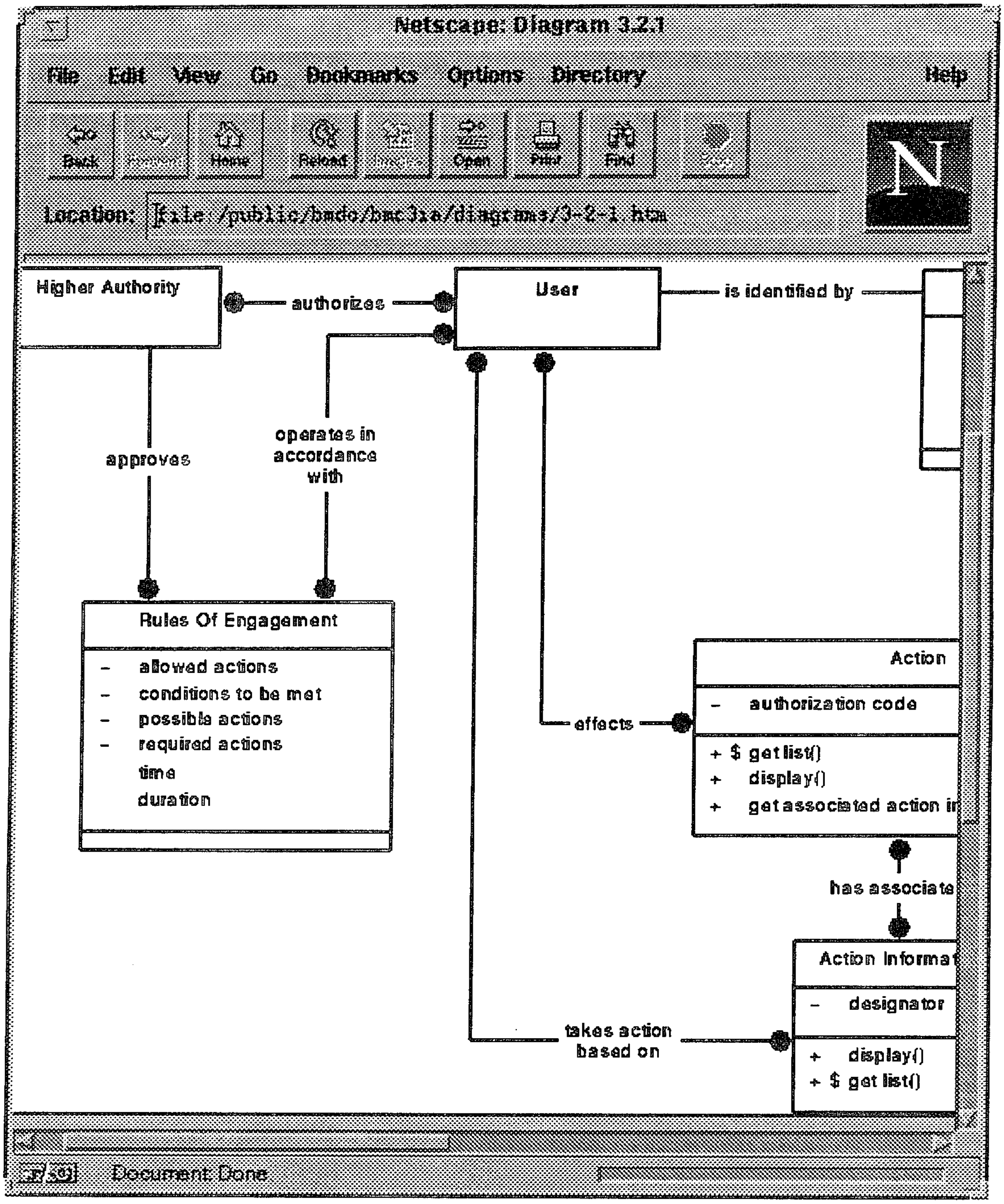

Figure 7. Screen Image: Sample Diagram

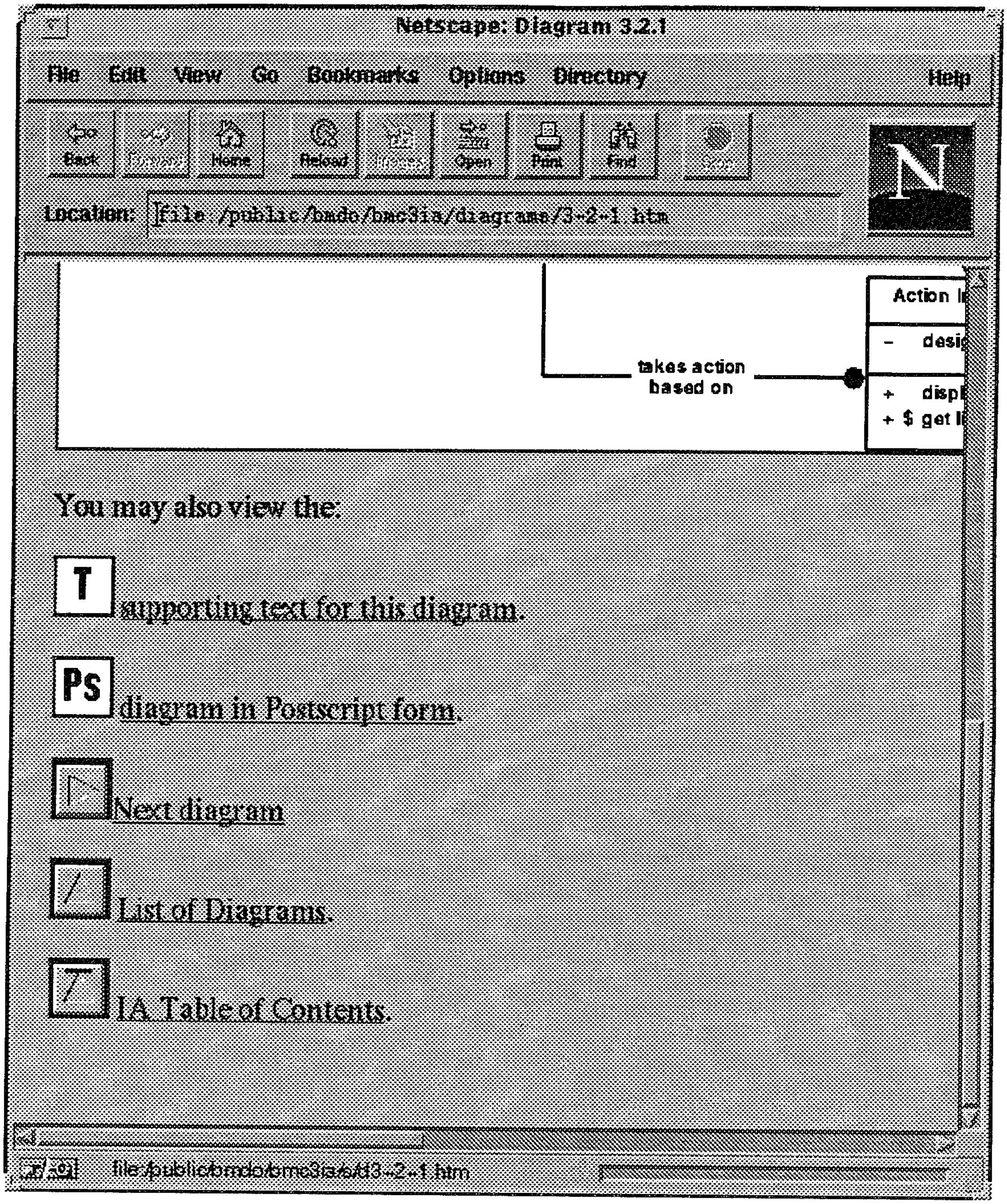

Figure 8. Screen Image: Pages Accessible from a Diagram

2.3 CLASS DESCRIPTIONS

The diagrams contain classes as well as their relationships. A class is an abstraction that describes properties important to the application; objects with the same data structure (attributes) and behavior (operations) are grouped into a class [Rumbaugh 1991]. A class description includes the class's name, attributes, operations, associations with other classes, and a list of the diagrams in which the class appears.

Most class descriptions reference other classes. Using the process described in Section 3.3, we developed programs that generated hypertext links to connect these references to the other class's descriptions. These programs also generated hypertext links from classes to all the diagrams in which those classes are contained. Figure 9 shows a screen image of a sample class description.

The original NMD BM/C3 IA stored class associations separately from the rest of the information about a class, but through use of initial versions of the Hypertext IA it was determined that in a hypertext document the class associations should be included with the rest of the information about that class. Our programs restructured the original document to do this.

A class dictionary is an alphabetized list of all classes. The Hypertext IA includes a class dictionary with links to each class description. The class dictionary is accessible from the IA table of contents, so if a user knows the name of a specific class the user can easily get to that specific class description using the class dictionary. The class dictionary is generated using our developed programs.

Many other types of information and cross-references were handled in a similar way: Special programs were used to extract information and create many hypertext links, permitting users to find information in a variety of ways.

Since a class description has links to all related class descriptions and to all diagrams that contain that class, a user can start at any point (say with a diagram or specific class) and, using hypertext, quickly navigate to all related information that interests the user. This makes it easy for the user to quickly examine a specific subset of information.

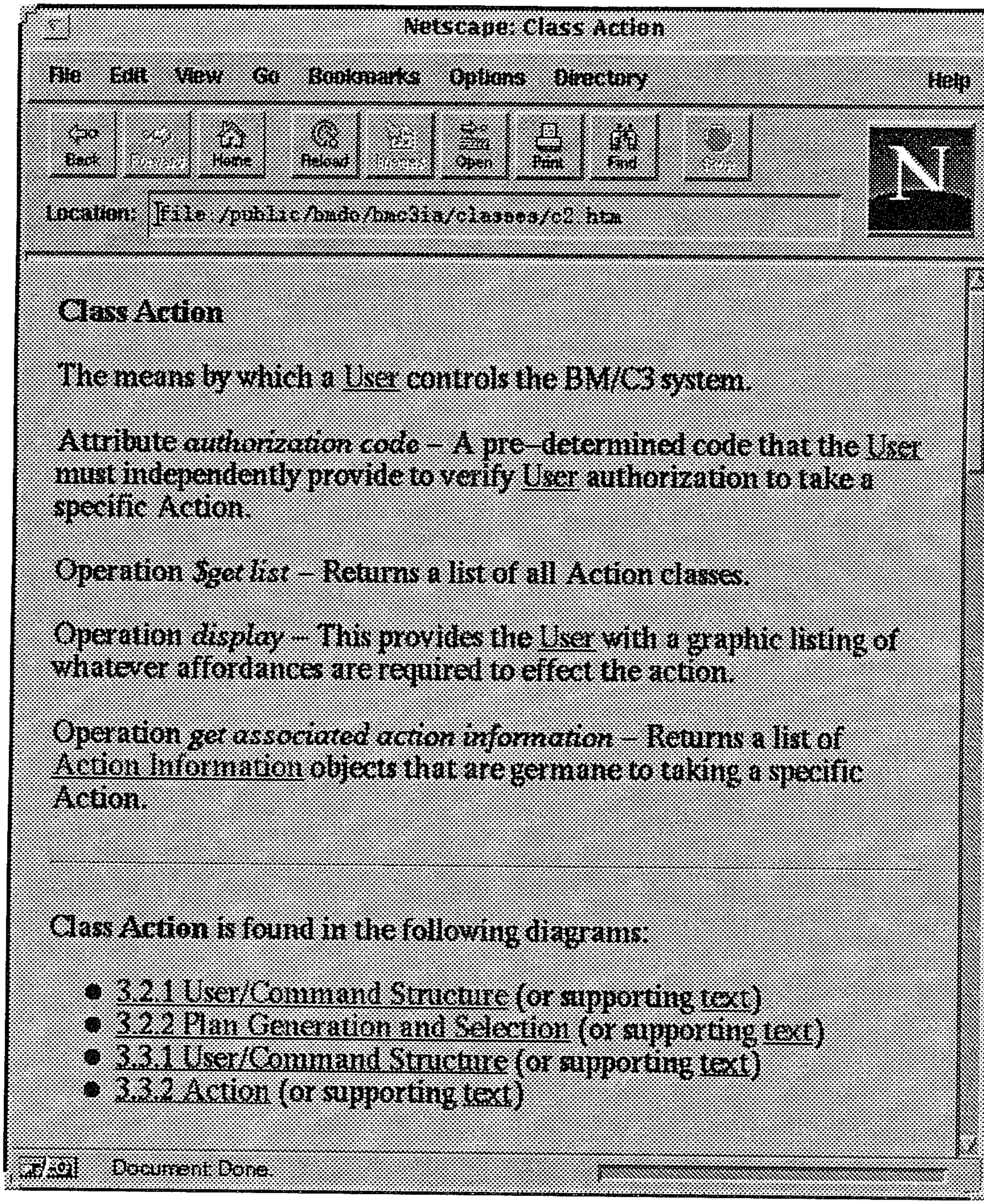

Figure 9. Screen Image: A Class Description

3. GENERIC PROCESS

This section describes a generic process for developing a tightly woven hypertext document from existing documents. Figure 10 shows the steps of this process. Note that this is an iterative process. For example, the desired results may change due to lessons learned from the implementation. Each step is described in greater detail in the following sections and includes examples based on our prototype implementation.

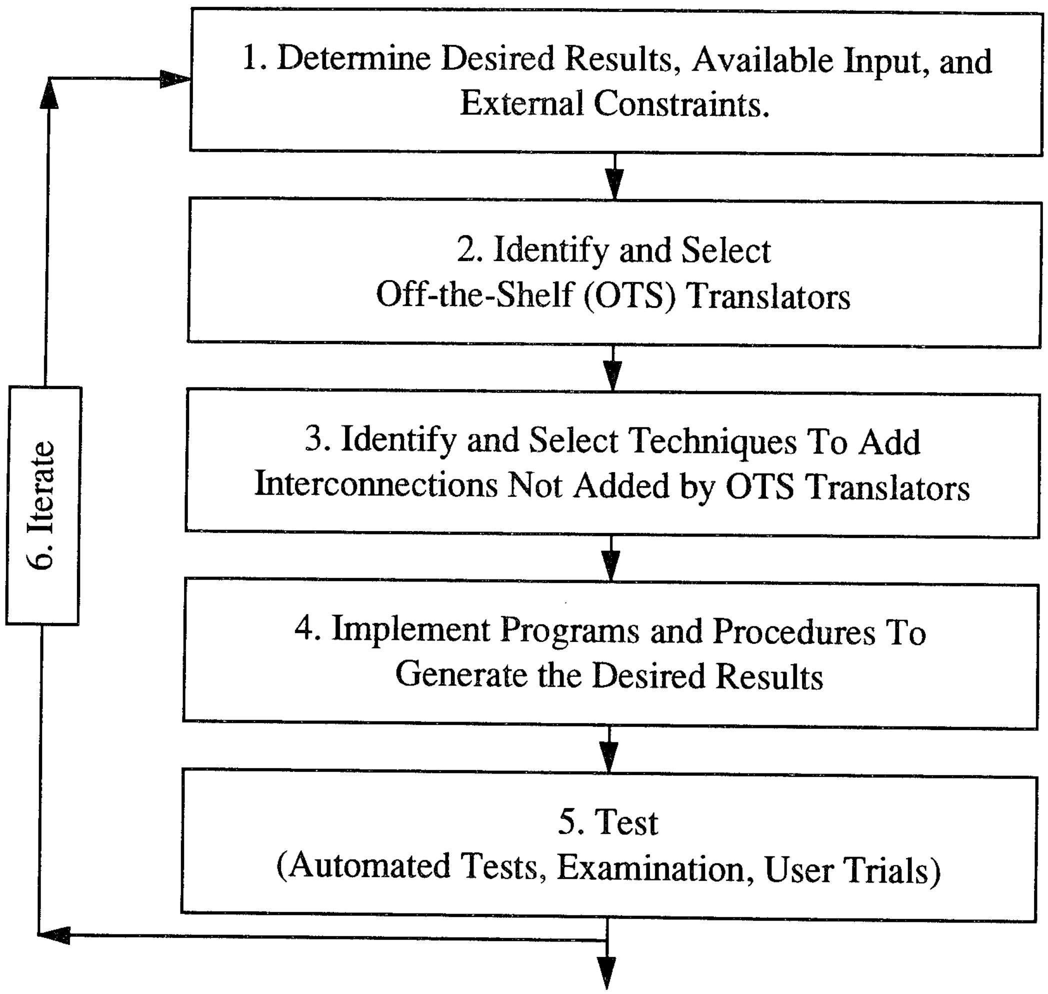

Figure 10. Generic Process Steps

3.1 DETERMINE DESIRED RESULTS, AVAILABLE INPUT, AND EXTERNAL CONSTRAINTS

The first step of any project is to determine the desired results given the available input and external constraints. In the following sections are questions we found important to answer along with a brief discussion. In some cases examples from the Hypertext IA are included.

3.1.1 Desired Results

a. *What is the expected use? What are the typical questions a user will ask? What are the related types of information that the user may want to see when viewing a given piece of information?* Based on our experience we determined that IA users would want to navigate from the graphical diagrams to the diagram descriptions, and from there to the more detailed descriptions of the classes in a given diagram. We also determined that users would want to navigate from class descriptions to any related class description and/or to any diagram that contained the class. Finally, we determined that users would want to be able to read the background material available, but would rarely visit it more than twice.

b. *What are the desired hypertext and graphic output formats?* Output formats should be selected that can support the expected use of the document. In addition, it is best to fix on a standard approach so other documents can benefit from an identical or similar approach. Therefore the selected output formats should also support foreseeable uses for future documents and be supported by many vendors. Low per-use cost is also an important consideration.

   For our prototype we selected HTML as the primary output format because HTML supports access over an internet (including the public Internet), the format is supported by many vendors, the format itself is not proprietary (so future multivendor support is likely), and HTML browsers are inexpensive or free. Other output formats with different strengths and weaknesses exist, such as specific word processor hypertext formats and groupware products. A comparison of different output formats is beyond the scope of this paper.

   As noted in Section 2.2, we did not find a single ideal graphical output format. We wanted to support whatever screen resolution is available, support zooming in and out, support "embedding" a graphic in a larger hypertext page, and be viewable by most users without additional tools. No single graphical format currently meets all of these requirements. The solution we reached was to provide two data formats for all important graphics: Adobe's Postscript and GIF. Postscript supports zooming in and out and is independent of the screen or printer resolution. GIF files can be embedded in hypertext pages and GIF viewers are built into most Web browsers.

c. *What will be the general organizational structure of the results?* For the prototype we determined that the original document's table of contents was a useful starting point for organizing the data.

d. *Is a tightly woven document desired?* If not, step 3 in Figure 10 is eliminated and step 4 is usually trivial. The answer to this question depends on the expected use. If users will want to navigate to a large number of related components from a given component, a tightly woven document will probably be desired.

e. *What will different iterations contain?* It is best to prioritize the desired results so that the most important items are developed first. For the Hypertext IA, it was relatively easy to do a first iteration using the table of contents and background sections, using placeholders as necessary, and then adding different types of interconnections to each iteration. The contents of each prototype iteration are further described in Section 3.6.

3.1.2 Available Input

a. *Can the input data formats and document conventions be selected or changed to ease generation of a tightly woven document?* The process of developing a tightly woven document can be made easier if the document developers are willing to select or change their data formats and document conventions to ease the process of generating a tightly woven document. Where possible, input data formats should be selected that are well supported by off-the-shelf (OTS) software. Document conventions should be selected that make it easy to identify link sources, link destinations, and matches of sources and destinations. For the Hypertext IA, such a selection or change was not possible because the source document already existed.

b. *What input data formats will be available?* The input data formats will determine what OTS software will be needed for step two. The input data format may also limit what can be done with the data since some formats are difficult to process for the purpose of developing a hypertext document. In many cases different portions of the data will be stored in different formats. In the case of the Hypertext IA, the input data were provided in Microsoft Word, Microsoft Excel, Adobe Postscript (generated by Interactive Development Environment's CASE tool Software through Pictures), and Macintosh Draw formats. Thus, for the prototype we needed to use tools and approaches that would handle these specific formats.

c. *What important conventions are followed in the input data?* Consistent input conventions can significantly ease the process of developing a tightly woven document. Particularly important are the conventions used to mark a link source or destination. For example, in the IA's class definition section every new class was marked with the text phrase "CLASS:", making the beginning of new class definitions (a link destination) easy to identify.

3.1.3 External Constraints

a. *What resources are available?* The amount of time, money, and personnel will determine the kind of obtainable results. Tools may need to be purchased, and personnel will then need to learn how to use them.

b. *How rapidly is the input data changing? What is changing?* If the input data changes often, a more automated approach may be better (so that changes can be quickly incorporated). The IA itself was changing while we were developing the hypertext version. Thus, we were driven toward an approach that could automatically generate the hypertext version so updates could be quickly incorporated.

c. *What computing environments must be supported?* This question leads to many other questions. What computers and operating systems must be supported? Will a server be available, or must the results stand alone? What is the bandwidth between the user and the data? (A wide area network usually has a smaller bandwidth than a local area network or local disk.) Will some users want a text-only interface (such as users with slow machines or low bandwidth connections)? Is it appropriate to require the user to install new software? And if not, what software does the user already have available?

For the Hypertext IA, we determined that supporting a wide variety of environments was necessary, and that it was not appropriate to require users to install a program called a Web server (which would have made some capabilities like on-line text searches possible at the expense of increased installation time). We also decided that requiring the installation of a general-purpose HTML browser was acceptable because they are inexpensive and easy to install, but that no other software should be required. We also determined that wide area access would be possible, so large graphics should only be loaded when required while small graphic files could be automatically loaded with their corresponding text.

3.2 IDENTIFY AND SELECT OTS TRANSLATORS

System development information is often stored in data formats different from the formats of the desired results, so translators may be needed. Translators are programs that convert one data format into another. Some translators are part of applications programs (such as word processors) while others are stand-alone products. While in some circumstances it may be necessary to develop a translator, in most circumstances there are OTS translators that can aid development of tightly woven documents.

Current OTS translators are aids but they do not create the tightly woven documents helpful for understanding complex, highly interrelated information. Current OTS translators create only a few hypertext links between the components of the information. Translators can create hypertext links for graphics embedded in text, and some can create links between sections of a document if those section divisions are tagged as section divisions in the original. Other processes may be necessary to complete the work, as described in Section 3.3.

To identify translators, search for one as you would search for any piece of software. The Internet is a particularly rich source for translators and information about them. Two sources we found useful were Yahoo's section on HTML converters[7] and the graphics file formats Frequently Asked Questions.[8]

Translators should be evaluated before using them because for most formats there are many different available translators with different strengths and weaknesses. Some translators occasionally mistranslate the data (e.g., some punctuation characters were erroneously translated into foreign language characters by one tool). Translators vary in their capabilities (e.g., some can only handle text and not graphics or tables). The translated

---

[7] At URL `http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/HTML_Converters/`

[8] At URL `http://www.cis.ohio-state.edu/hypertext/faq/usenet/graphics/fileformats-faq/faq.html`

results may be difficult to use and/or display poorly, depending on the quality of the translator. Some translators are more flexible and permit users to control the translation process. Price is also an important consideration. Some problems can be overcome by using additional programs.

To select a translator, examine the documentation of candidate translators and try out samples of real documents to determine which candidate translators are acceptable. In particular, check for features that are important for processing your documents but that are not supported by the translator. If your source information has graphics, tables, or equations, make sure that the translator can handle them. Identify necessary hypertext links not added by the translator.

Often translating to intermediate formats can improve the results. For example, at the time we developed the Hypertext IA, there was a beta version of Microsoft Word that could generate HTML. We obtained better results by first translating the document to another data format, Microsoft's Rich Text Format (RTF), and then running a separate program that translated RTF into HTML.

Translators are a particularly volatile software area; what was the best last year may not be the best this year, and data formats are often changed and new ones created. Therefore, the decision of what translators to use should be re-evaluated periodically. Later stages of the development process should be developed so that they are less dependent on a particular translation tool.

3.3 IDENTIFY AND SELECT TECHNIQUES TO ADD INTERCONNECTIONS NOT ADDED BY OTS TRANSLATORS

OTS translators do not add all of the hypertext links desired for a tightly woven document, so a process to examine the translator results and add additional hypertext links is necessary to create tightly woven documents. We developed the following process to add these interconnections:

a. *Identify desired hypertext links that were not generated by the translators.* This identification is based on the results of the previous two steps. Documents translated from word processors typically have few hypertext links, but documents generated from CASE tools could conceivably include many of the desired hypertext links.

b. *Identify hypertext link destinations, preferably in terms of general rules.* Instead of manually identifying each link destination, it is often best to find general rules for identifying destination links so changes can be handled automatically. Often link destinations are easily identified in the text as section headings, specially formatted text, or text marked with special patterns. For example, the IA diagrams were marked as section headings in one particular chapter, and the class descriptions were marked using the phrase "CLASS:" in another section.

c. *Identify hypertext link sources, preferably in terms of general rules.* Link sources are the locations that users can select to navigate to a link destination. As with link destinations, it is usually best to find general rules for identifying link sources so changes can be handled automatically. In some cases these hypertext link sources may be identified using special formatting. For example, IA references to class names always used initial capital letters (e.g., Rules Of Engagement). Unfortunately, link sources may not be identified as easily as destinations. For example, in the original IA, initially capitalized words are not necessarily link sources—every sentence begins with an initial capital letter.

d. *Identify which types of links should be generated automatically and the basic techniques for doing so.* One useful technique is to first process the document to generate a list of all link destinations, then go back and locate matches to those link destinations (considering these matches the corresponding link sources). For example, in the Hypertext IA, a list of all classes was created by processing the document. The document then underwent a second pass, and any time the name of a class was used in the description of a diagram that name was considered to be a link to the matching description of that class. Identify which locations in the document these connections are appropriate; they may not be appropriate in all locations in the document.

e. *Identify synonyms and likely input mistakes.* The singular and plural forms of a word (e.g., Rules of Engagement vs. Rule of Engagement) may link to the same destination. Descriptions may use variations in phrasing (e.g., Engagement Rules). Some variation in spelling, capitalization, or format may be present (e.g., "Rules Of Engagement" vs. "Rules of Engagement"). For all of these reasons, a "synonyms" list is useful; such a list maps alternative phrases into "standard" phrases for the purpose of translating a document into hypertext. Note likely input mistakes that could affect the results, such as incorrect capitalization of links and missing table entries, so they can also be handled.

f. *Identify which links must be created manually and the approach to creating them.* If no general rule can be found for generating a link, the specific link source and destination can be specifically identified and created manually.

There are two kinds of manual links: *persistent* and *impersistent.* Persistent manual links are retained in a tightly woven document when the document is modified; impersistent manual links must be manually recreated every time the tightly woven document is generated (say, after a change in the information). The persistence of a manual link is determined by the process used to generate the manual link. If humans edit the output of the translators to add manual links, the links are impersistent—the next time the tightly woven document is generated, the results would need to be re-edited. If humans add links to the source data (say, by using a word processor that supports hypertext links), these manual links will be persistent.

Persistent manual links are preferred over impersistent links. If a document undergoes many revisions, impersistent manual links can make generating a tightly woven document more difficult, time consuming, and expensive than persistent manual links or automatically generated links. If the original data source does not permit adding manual hypertext links, small programs can be used to add manual links so the manual links will be persistent.

If a large number of links must be created manually, it may be too time consuming to add them. This circumstance suggests that the approach, desired results, input format, and/or input conventions need to be changed.

3.4 IMPLEMENT PROGRAMS AND PROCEDURES TO GENERATE THE DESIRED RESULTS

The following actions should be performed to develop the programs and procedures to generate the desired results. These programs and procedures implement the requirements identified by steps 1 and 3.

a. Select the programming languages and tools to help generate the interconnections. Automatically generating hypertext links is primarily a text processing task, so text processing languages are likely to be useful. Word processor macro languages and CASE tool report languages may also be useful. We selected the text processing language Perl because it was free, widely supported, and included many useful text processing capabilities [Wall 1992, Schwartz 1994].

b. Determine the high-level approach for performing the automated tasks.

   1. For the Hypertext IA we decided to first run all OTS translators and then perform special processing on the translator results. The top half of Figure 11 shows the resulting process used to generate the Hypertext IA. We developed the programs to implement the special processing.

   2. We decided to implement the special processing step using a set of many small processes built from a set of small scripts. Each script was modular, i.e., it performed a specific function and had a clearly defined interface. Most scripts were designed to work in a pipeline. All scripts were placed in a common library so they could be reused. A common programming utility, "make," was used to combine these small scripts together and determine which processes needed to be executed in which order. Using many small pipelined scripts from a single library made it easy to reuse and recombine scripts to change the results. We also expect that this approach would make it easier to generate additional tightly woven documents.

      One example of a special processing process is shown in the bottom half of Figure 11; this example shows the process used to create the Hypertext IA class descriptions and the class data dictionary. A preprocessor was first executed to make later processing easier. The classes were then identified and two different outputs resulted: a list of classes (which was used to generate the class dictionary) and the actual class descriptions. The script "addclink" processed the class descriptions, adding links from references to class names to the class definitions. Not shown is the process to identify the destination links and to handle synonyms, which generates data used by "addclink.".

c. Before processing the input data with other scripts, preprocess it with a special preprocessing script. Doing so has the following advantages:

   1. OTS translator errors and omissions can be easily fixed in one place. For example, the translator to HTML we used translated incorrectly certain punctuation characters into foreign language characters.

   2. Changes to the OTS translator (including updates and changes to a different tool) can be handled by the preprocessor. The preprocessor can ensure that the data as seen by the specialized scripts has a simple, standardized format.

3. Such a preprocessor can make other scripts easier to create. For example, in the Hypertext IA the preprocessor grouped entire paragraphs into a single text line so that later operations (which performed substitute and replace operations) did not need to handle text line breaks.

d. Emphasize reuse. Defining general-purpose parameterized modules results in a system that is more evolvable. Reusable modules can also significantly reduce the cost of generating other tightly woven documents. A number of the scripts developed in this task (such as "preproc" and "fanout" in Figure 11) should be reusable in generating other hypertext documents.

e. Handle differences from the "standard" format by attempting to generate a reasonable result and sending warning messages. Likely differences from the "standard" format include upper and lower case differences and empty entries in tables. For example, in the Hypertext IA, classes were to be identified with the text "CLASS:", but one was identified with "Class:" instead, so the patterns used in special processing took this variation into account. As another example, a cross-reference table was not supposed to have empty rows nor columns, but the tools checked for this case and found some empty rows and an empty column. In this case, warning messages were displayed so the original data could be analyzed.

f. Consider using "intermediate" data formats. Using intermediate data formats can help break a task into meaningful smaller tasks and can make it easier to change things later. If no existing intermediate format is helpful, consider if creating intermediate data formats would improve the automated task design.

g. Identify ambiguous and multiple links, then determine how to handle them. If a given phrase should be a source link to multiple destination links, determine what should be done. If the phrase should actually link to multiple destination links and the hypertext system supports multiple links to a destination, then this is not a problem. If the hypertext data format does not support multiple links to a destination (e.g., HTML does not), then an intermediate page must be created to list the possible destination links. If the phrase should actually link to only a single destination, then some sort of prioritization scheme will be needed.

   For example, in the IA, many class references are textually ambiguous, e.g., "Rules of Engagement" could link to both class "Rules of Engagement" and class "Rules" if both classes existed. We chose to prioritize class name links based on word length; class names with more words are preferred over class names with fewer words, and once a choice is made no other link from that phrase is made.

h. Determine procedures for tasks that will not be performed automatically and then implement those procedures. For example, we decided to generate a first version of the table of contents using the OTS translator and then used an ordinary text editor to make the changes we desired.

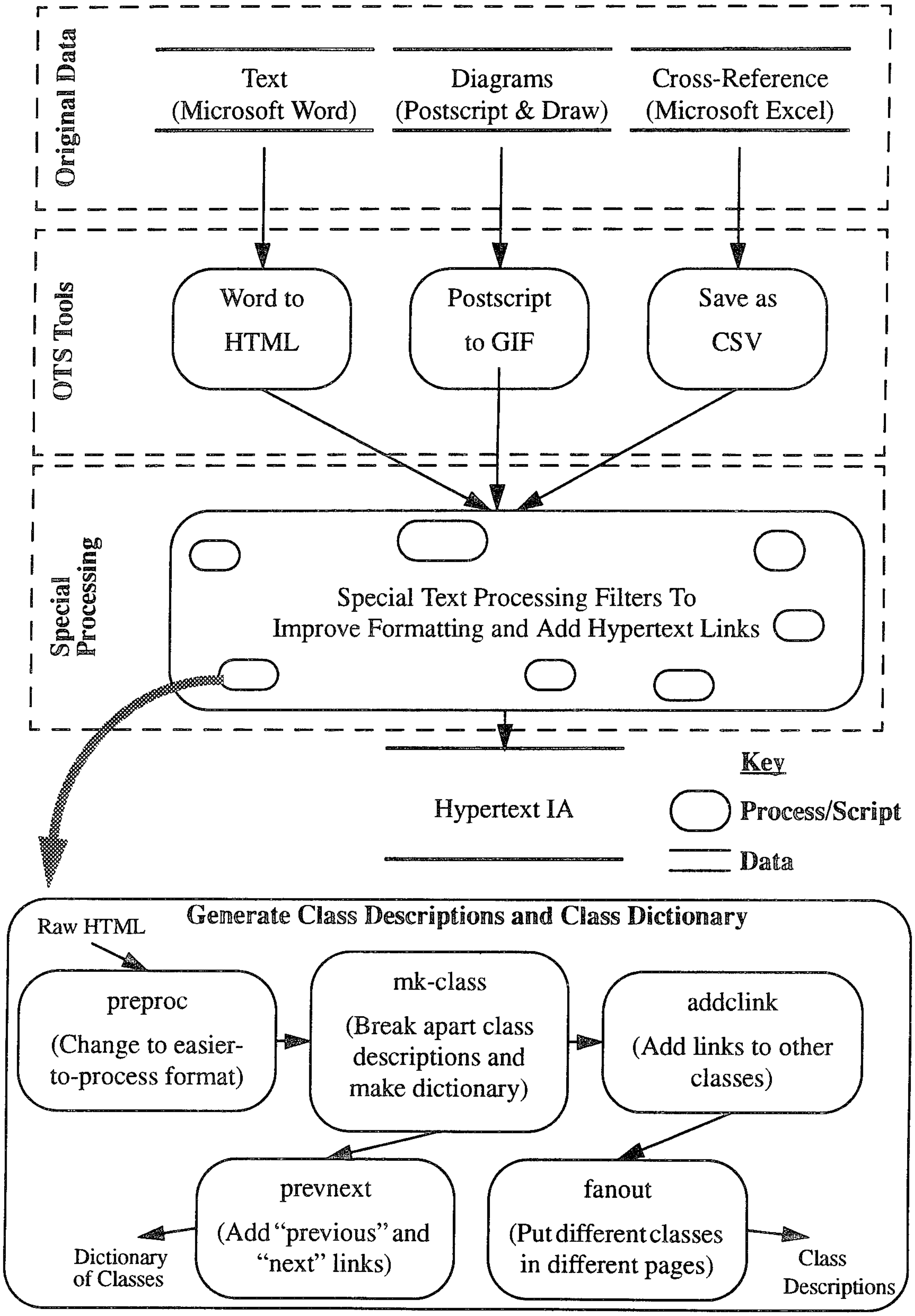

Figure 11. Hypertext IA Translation Process

3.5 TEST

We found three different kinds of testing useful:

a. Automated tests. Programs can check for a number of common mistakes, such as incorrectly formatted output and hypertext links with no destination. Using automated tools is important because it is impractical to manually check every hypertext link in a tightly woven version of a real system development document. We used weblint, an OTS program, to check for these common mistakes.[9]

b. Examination. Developers can often find problems by examining the results the same way a user would examine them. Besides determining if the result is what was intended, developers can ask the following questions:

   1. Are there standard links (such as previous, next, and up links) where appropriate?
   2. Are the graphics correct and legible?
   3. Is the performance acceptable (this is particularly an issue with large files)?
   4. Do the added hypertext links add value to the result?
   5. Are hypertext links missing that should be added?
   6. Is there another structure or format that would make the content easier to understand or use?

   If a script has been changed to produce a different result, examine the actual results to determine if the changes were correctly implemented.

c. User trials. Trials (also called "beta tests") by users and surrogate users are very helpful for identifying problems and improvements. Before the users evaluate the product, make sure they understand what is and is not in the product they are evaluating.

---

[9] Weblint may be downloaded from `ftp://ftp.khoral.com/pub/perl/www/weblint.zip`

3.6 ITERATE

If the results match the desired results, there is no need to iterate. However, we found that while developing a hypertext version of a document, our desired results changed. This was particularly true when the surrogate users began to use the results. This is normal in any user-interface-intensive system, and developers should plan to go through at least several iterations. Planned iterations also reduced risk; if project resources change or the effort was underestimated, some (incomplete) results would be available.

As an example, here were the Hypertext IA iterations:

a. The first version contained the basic table of contents, all the document sections as HTML files, one graphic in GIF format (to determine how graphics files would look), and links from class descriptions to other class descriptions. All class descriptions were in one file, with all class association descriptions in another file, mimicking the organization of the original document because this made it easier to quickly experiment with the results.

b. The second version added links from diagrams to class descriptions, and class descriptions were split into one class per hypertext page. Class association text was merged into the class descriptions. This improved the utility of the hypertext document and made its organization different than the paper document. Next, previous, and up links were added where appropriate.

c. Links from class descriptions back to diagrams were added, and all of the diagrams were converted into GIF format so they could be viewed.

d. Various usability enhancements were added based on user experience. These included additional special links between related sections and improving the visual presentation.

4. LESSONS LEARNED

The lessons we have from developing the prototype Hypertext IA can be divided into two categories: lessons learned about how to use the technology, and lessons learned about the technology itself (in particular the limitations of current technology).

4.1 HOW TO USE THE TECHNOLOGY

The following lessons were learned on how to use hypertext technology:

Where possible, plan for generating a tightly woven document before developing the original document. If the original document was intentionally developed to be used as a tightly woven document, it should be easier to generate a tightly woven document. Such planning should identify consistent formatting specifically designed to support generating a tightly woven document. The planning could also identify and configure tools to make generating tightly woven documents easy. Examples of such tools include word processor macro languages, CASE tool templates, and CASE tool report generation languages. The Hypertext IA was created without the advantage of this advance planning.

Require consistent formatting in the original information. Creating a hypertext version of a document is much simpler when the original follows a set of consistent rules for identifying link sources and destinations. As an example of consistent formatting, in the original IA all class names had initial capital letters when used as link sources, and class definitions (which were used as link destinations) were always preceded with the phrase "CLASS:". This consistent formatting made it much easier to create hypertext links in the hypertext IA. Consistent formatting may be difficult to enforce; if not done, this could significantly increase the amount of time necessary to develop a tightly woven document.

Plan for human-generated information to have errors and occasional inconsistencies. The tools to generate the Hypertext IA performed cross-checks of information to detect errors and inconsistencies, which helped to identify some problems. Some tools had to be made insensitive to upper case letters vs. lower case (e.g., "CLASS" vs. "Class") because human inputs occasionally do not follow the standard format. A separate "synonym" file was created to list synonyms since variations of a class name were used in descriptions.

Use tools to detect errors. It is easy to write scripts that generate "almost correct" or "mostly correct" hypertext documents. It is not practical to check every document and link by hand, e.g., the Hypertext IA has more than 8,000 internal links. As a result, it is important to use tools such as weblint to check the validity of the results. The original documents also often contain errors; tools should be robust in the face of these errors and report errors if practical to do so. Such errors might include incorrect data formatting and missing information.

Design for information updates. One approach to developing a hypertext document from original documents is to translate it "by hand" using a hypertext editor or text editor, but this is expensive to do and expensive to update. If the translation is done "by hand," the hypertext system would be out of date with the original data whenever the original is changed. Generating the hypertext document automatically makes it significantly easier to update the information. Automatic generation requires a great deal of text processing, so a text processing language is likely to be a useful tool. In a few cases automatic generation may be too difficult—for example, the Hypertext IA's table of contents is actually generated manually, but it is only one of 256 hypertext pages in the Hypertext IA.

Support both the novice and experienced user. For example, provide multiple starting points for both the novice and experienced user so experienced users can skip unnecessary material. Providing "help pages" and introductory material is helpful for the novice.

Ask for specific data formats. We did not ask for any specific data formats, so we received formats convenient for the data producers (in this case Microsoft Word, Microsoft Excel, and Postscript files; the Postscript files had been generated by a CASE tool). While these formats did not cause us significant technical problems, the process of converting these into Web formats (such as HTML) was time consuming. Different computers had to be used to run different OTS tools, requiring file transfers between different machines. Some of the conversion tools did not support a scripting capability and so had to be manually commanded to translate each file. It would have been easier for us if the information had been translated by the developer to specific formats (such as HTML) ahead of time.

Break information into the amount of one to three paper pages unless the information must be read sequentially. Breaking information into sizes that can be presented in one to three paper pages reduces the retrieval time and the probability that the user will be confused by extraneous information. Sections of text meant to be read linearly should be kept intact, however, because that makes it easier to read and consider as a unit. For example, the IA executive overview and introduction were kept intact (even though they were longer) because only the linear order made much sense. The rest of the document was split into units one to three paper pages long.

Provide basic document navigation operations in every document. It is important to include a "table of contents" to make it easy to get to any specific place in the document from a central point. Also include "next," "previous," and "up" buttons where appropriate.

Use the original document organization as a starting point for hypertext information organization and then vary it as appropriate. Once we had developed an initial version of the hypertext document, it was easier to see how to reorganize the information to improve the hypertext document's quality:

a. Certain kinds of information belonged together that were separate in the original. For example, class descriptions and associations were in separate parts of the document, but they made more sense when combined in the hypertext documents.

b. Some other items were split apart because putting them together made the document too slow to use. For example, diagram graphics were slow to load, so the supporting diagram text was made separate from the graphics so people could read the text without loading the graphics. Originally, all of the classes were considered a single document, but it was better to create a large number of documents, each one describing a single class.

Develop hypertext systems iteratively. Our approach was to develop a basic hypertext system and then go back and improve it ("learn by doing"). For systems which are user-interface intensive, such as a hypertext system, this is usually an excellent way to build a system. It was faster to develop a usable version and receive feedback than delay until a "perfect" version was available.

Many skills and domain knowledge areas are useful for setting up to automatically generate tightly woven documents. Some of the useful computing knowledge areas include the planned hypertext output format (e.g., HTML), text processing languages, and user interface design. Understanding the document's domain (e.g., missile domain) is useful for identifying appropriate links.

Hypertext documents are only as good as their information source. While this should be obvious, it bears repeating: Hypertext can only make it easier to navigate information, it cannot overcome problems with the source information (such as ambiguous, incorrect, incoherent, or missing information).

Tightly woven documents are practical and useful. Documents with a large number of links from any given page can have significant benefits for documents with a large number of logical internal interconnections.

4.2 CURRENT LIMITATIONS OF THE TECHNOLOGY

Current technology is sufficient for the task, but there are limitations and difficulties that will probably be removed within the next few years. Most of the limitations and difficulties involved the Web technology. Overall, we found that Web technology is sufficient for the task and that it provides significant new capabilities. However, we also found limitations and ease-of-use problems with Web technology due to its maturing nature. We expect many of these difficulties to be removed soon since this is an area that is rapidly improving. The following are specific lessons learned about the technologies we used; current users must account for these limitations when building systems today.

Current Web document formats are limiting. In particular:

a. Graphics can be in GIF bitmap form which does not adequately support zooming, or in Postscript which does not support internal hypertext links and is not supported as an in-line graphic format by most browsers. Our solution to this problem was to provide multiple graphic formats, each with its own advantages.

b. It is very time consuming and excessively complex to create links inside a graphic when there are many such links. Currently, to create hypertext links from a graphic, someone must view the graphic and identify each rectagular area in the graphic and what it links to. Current tools can make it is easier to identify and record these areas, but this becomes labor intensive when defining hundreds of such areas that may need to be redone on new versions of a graphic. Such links may also require additional installation and administration time; for most browsers these links require support from a separate program (a Web server). A better approach would be for the tools that generate the graphics to simultaneously generate all of the hypertext links or at least identify the various link sources in the graphic.

c. HTML, the standard format for Web text, provides little control over formatting. The version of HTML that is widely available at this time (HTML 2.0) does not even support basic formatting capabilities such as paragraph indentation and superscripts, reducing the quality of the results. HTML does not currently support style sheets or other techniques for finer control of document formatting.

Some word processors and desk top publishing programs do not directly support Web technology. This is expected to change quickly. The latest versions of Microsoft Word and Adobe Corporation's FrameMaker have added Web support such as automatically generating the HTML format and supporting hypertext links in the document itself. This support eases the generation of tightly woven documents. Lack of such support in these types of applications is not a major problem since there are stand-alone tools that can convert many document formats into Web formats.

Graphics are slow to display on some systems using Web technology. It is best to keep graphics small or to provide graphics as separate files linked to the document when using Web technology because it may take a long time to display them. Where possible, the document should be usable without graphics to support users with especially slow graphics display speed. Three factors significantly influence graphic display speed:

a. Slow hardware/software systems. For example, a 33Mhz 80486 IBM Thinkpad running Netscape 1.1 and Microsoft Windows 3.1 took 30 to 50 seconds to display the larger diagram graphics stored on its local disk. This seemed unacceptably slow; a delay of three to five seconds is more commonly desired. Faster computer hardware and more mature software should rapidly improve the display speed of hardware/software systems.

b. Network bandwidth and utilization. Graphic files can be large, and downloading them over a network (such as the Internet) can greatly increase the display time, depending on the bandwidth and utilization of the network. For local area networks this is usually not a significant problem, but for wide area networks (such as the Internet) this can be a primary factor affecting display speed.

c. Caches. A cache stores copies of recently retrieved information so they can be retrieved again quickly; larger caches can store more information than smaller caches. If a specific graphic has been displayed recently, it can be redisplayed quickly using a cache. HTML browsers and other software can be configured to support large caches which may improve graphic display speed.

Text processing languages are widely available, but improvements are desirable. Text processing languages ease the task of developing scripts to generate tightly woven documents. We selected the text processing language Perl, a widely used text processing language. Perl provides a large number of string manipulation, regular expression handling, file processing, and table lookup functions, all of which were useful for automatically generating a hypertext document. A significant problem is Perl's excessive permissiveness—Perl assumes that the "programmer is always right." Perl detects very few programming errors (compared to most general-purpose production languages) and has an error-prone syntax. As a result, a significant amount of development time is unnecessarily spent as debugging time.

In spite of these limitations, hypertext systems can be developed that are useful and worth producing, and we expect these limitations to be eliminated within the next few years.

5. RESULTS AND DISCUSSION

The experience reported here on transforming the NMD BM/C3 IA into a prototype tightly woven document appears to have met its goals. As outlined in Section 1.3.3, these specific goals are to:

a. Develop a tightly woven version of the NMD BM/C3 IA.

b. Handle document updates quickly and at low cost (due to changes in the NMD BM/C3 IA). This implies that the generation process must be highly automated since manual regeneration of an entire document after each revision would take too long.

c. Require a small per-user cost. CASE tools are typically quite expensive ($10,000 or more per user), significantly reducing the number of potential users. A per-user cost of $50 or less for any necessary tools permits more users to access the information.

d. Demonstrate the utility of tightly woven documents for system developers to navigate complex system development documents.

e. Aid in identifying techniques, processes, and lessons learned for developing tightly woven documents.

The following subsections give evidence that each of these goals have been satisfied. This evidence suggests that other documents developed using the same process may also meet these goals.

5.1 TIGHTLY WOVEN INFORMATION ARCHITECTURE

The Hypertext IA contains more than 8,430 internal hypertext links between 256 pages (nodes). Thus, the mean number of hypertext links per page is 8,430/256 = 32.9 links per page. The median is 15.5 links per page. These values are far more than the typical three to four links per page generated by simple hypertext generators (which generally have a link to the next page, previous page, the table of contents, and a miscellaneous link to a graphic or footnote). This large number of links per page suggests that the Hypertext IA is tightly woven.

5.2 AUTOMATICALLY GENERATED DOCUMENT UPDATES

The Hypertext IA is generated almost entirely automatically. Of the 8,430 links, only 162 links are generated manually (3 special cases, 63 inter-section links, and 96 links from the manually updated table of contents). This means that 98% of the links are created automatically. Of the 256 pages, 255 (99.6%) are generated automatically using approximately 30 scripts (containing approximately 1,300 noncomment lines) and five OTS packages.

5.3 SMALL PER-USER COST

The Hypertext IA is accessible using a low-cost Web browser (costing from zero to $49), not an expensive CASE tool (often costing $10,000 to $25,000). Compared to CASE tools the per-user cost is very low.

5.4 HYPERTEXT IA UTILITY

To obtain some initial user feedback, four IDA staff members were asked to evaluate the Hypertext IA and provide comments via electronic mail. One staff member used the Hypertext IA as part of his work to evaluate another BMDO NMD product (the executable representation). All used the IA as typical users would, i.e., moving between specific diagrams and various kinds of detailed text to gain understanding of specific areas of the system. Most of the comments were minor formatting recommendations; overall, these surrogate users found the Hypertext IA straightforward and easy to use. Here are some sample comments:[10]

> Generally, I think the hypertext IA is a nice piece of work... [other information] would be great in this form. Chasing hypertext links is much less painful than [using computer text search commands to find related information].
>
> I found myself forgetting where I saw certain statements. Hypertext does seem to help you pop back to several likely places to find what you remember. Hypertext aids navigation but not necessarily understanding, which is mainly affected by the expressed content and the user's capability with aids to comprehend all he/she "touches" of the content. A companion aid that is well-needed is a note-taking tool, such as the text editor that I brought up in another window as I read the hypertext product and used to make these notes.

---

[10] While we normally avoid quoting other co-workers, quoting them as surrogate users seems appropriate.

It is difficult to categorize the minor formatting recommendations; they involved such things as setting better titles, generating higher-resolution graphics to make text more legible, and creating "short forms" of some long documents.

The electronic diagrams were in some ways easier to use than the paper version. The electronic diagrams permitted users to use scrollbars to view different portions and to zoom in and out of the complex diagrams. In the paper document, the diagrams were available in one-page versions that could not be read (due to the tiny fonts used to fit the information on one page) and multi-page versions that were hard to use.

In short, the surrogate users found that the tightly woven document was useful; navigating the information was much easier using the tightly woven document instead of the paper document.

User comments from several presentations have also been generally positive.

5.5 TECHNIQUES, PROCESSES, AND LESSONS LEARNED

The prototype was an aid to identifying techniques, processes, and lessons learned for developing tightly woven documents. In particular, by developing this prototype:

a. A generic process and specific techniques for generating tightly woven documents were developed and demonstrated. The generic process and specific techniques to support this process were discussed in Section 3.

b. A library of software modules was developed; some of these modules should be reusable by future tasks. Some of these modules were discussed in Section 3.

c. Lessons learned were identified and were discussed in Section 4.

We believe this process is repeatable and would be progressively easier to perform.

Based on our experience gained from developing the prototype and use of the prototype, we believe that tightly woven documents can be worth the time required to develop them.

6. CONCLUSIONS AND RECOMMENDATIONS

Complex documents with many internal interrelationships, such as many system design documents, can be made easier to navigate using hypertext to create tightly woven documents. Tightly woven documents are hypertext documents with a large number of internal hypertext interconnections. These interconnections help the user to understand the interconnections of the system being described and permit navigation to whatever information is important to the user.

This paper presented a generic process for generating tightly woven documents. If the original documents were intentionally developed to be used as tightly woven documents, this process should be easier to perform, but the process also applies to existing documents where no such intent existed. This process can be summarized as the following steps:

a. Determine desired results, available input, and external constraints.

b. Identify and select off-the-shelf (OTS) translation tools.

c. Identify and select techniques to add interconnections not added by OTS translation tools.

d. Implement programs and procedures to generate the desired results.

e. Test.

f. Iterate.

New tools, such as HTML browsers, are now available that can significantly reduce the cost of developing and distributing system development information as tightly woven documents. There are limitations with Web technology, like any other new technology, but the technology is sufficient to the task and many of these limitations will probably be eliminated over the next few years.

We recommend that BMDO consider, at document conception, developing and distributing key system development documents as tightly woven hypertext documents using the information discussed in this paper.

APPENDIX A. SAMPLE SCRIPT

This appendix includes one of the many scripts used in generating the Hypertext IA as an example of a script. The script in this appendix is "preproc," a short script that takes as input an HTML file and generates as output a corresponding HTML file in a canonical format. This canonical format eliminates errors performed by the translator to HTML. It also makes later processing easier by grouping each paragraph into a single text line and by placing HTML commands in specific location. By placing the information in a canonical format, other tools are isolated from changes in the translator used to generate the HTML files.

This script is written in Perl [Wall 1992, Schwartz 1994].

```
#!/usr/local/bin/perl -w
# preproc - preprocess an HTML file to give it a "canonical" format
#           and fix any global problems with the translator to HTML.
# input:  an HTML file.
# output: an HTML file in canonical form.

# The canonical HTML format has the following form:
# * Each HTML paragraph is a single very long text line
#   (so global substitute and replace commands that work on a line at a time
#   won't miss phrases split on text line boundaries).
# * Useless formatting commands (like bold followed immediately by unbold)
#   are removed.
# * Certain HTML commands are forced to begin a line, end a line, or be on
#   a line by themselves. Standardizing the placement of these commands
#   makes later processing much easier.  In particular:
#   + The following HTML commands are placed on a line by themselves:
#      <P> <HTML> </HTML> <HEAD> </HEAD> <BODY> </BODY> <PRE> </PRE> <TR> </TR>
#   + The following HTML commands always begin a new line:
#      <Hn> <TITLE> <LI> <!-- comment -->
#   + The following HTML commands always end a line:
#      </Hn> </TITLE>
# The canonical HTML format doesn't force HTML commands to be upper or lower
# case, since that's pretty easy to handle.  If that's desired, change
# this program to make HTML commands the desired case.

# Developed by David A. Wheeler.
# (C) 1995 Institute for Defense Analyses.
# The Government of the United States is granted an unlimited license to
# reproduce this software.
```

```
# DISCLAIMER OF WARRANTY AND LIABILITY
# THIS IS EXPERIMENTAL PROTOTYPE SOFTWARE. IT IS PROVIDED "AS IS" WITHOUT
# WARRANTY OR REPRESENTATION OF ANY KIND. THE INSTITUTE FOR DEFENSE ANALYSES
# (IDA) DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING
# THIS SOFTWARE WITH RESPECT TO CORRECTNESS, ACCURACY, RELIABILITY,
# MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR OTHERWISE.
#
# USERS ASSUME ALL RISKS IN USING THIS SOFTWARE. NEITHER IDA NOR ANYONE ELSE
# INVOLVED IN THE CREATION, PRODUCTION, OR DISTRIBUTION OF THIS SOFTWARE
# SHALL BE LIABLE FOR ANY DAMAGE, INJURY, OR LOSS RESULTING FROM ITS USE,
# WHETHER SUCH DAMAGE, INJURY, OR LOSS IS CHARACTERIZED AS DIRECT,
# INDIRECT, CONSEQUENTIAL, INCIDENTAL, SPECIAL, OR OTHERWISE.


$previous_line_had_no_separator = 0; # if 1, we need a separator.
$in_pre_section = 0;                 # if 1, we're in a <PRE>..</PRE> section

while (<>)           # Read in each line and process it.
{
 chomp;              # Eliminate end-of-line characters.
 s/[\r\n]*$//;

 s/<b><\/b>//ig;     # Eliminate useless "bold" commands.
 s/<b>\s+<\/b>/ /ig; # Eliminate useless "bold" commands with spaces inside.
 s/<i><\/i>//ig;     # Eliminate useless "italic" commands.
 s/<i>\s+<\/i>/ /ig; # Eliminate useless "italic" commands with spaces inside.
 s/<tt>//ig;         # Eliminate tt commands (they're inappropriate)
 s/<\/tt>//ig;       # Eliminate end-tt commands (same reason).

 # The translator to HTML gets the following translations wrong,
 # so fix them here.
 s/\[\[yen\]\]/*/g;  # Replace "yen" with a readable character.
 s/\[\[Otilde\]\]/'/g;
 s/\&Ograve;/"/g;
 s/\&Oacute;/"/g;
 s/\&#142;/&eacute;/g;  # acute e, e.g., "applique'" and "resume'".

 # Now break lines up as needed.
 # From here on we're inserting newline characters into the current line.

 # Line by themselves: <P>, <[/]HTML>, <[/]HEAD>, <[/]BODY>, <[/]PRE>, </TR*>
 s/<P[^>]*>/\n$&\n/ig;
 s/<\/?HTML[^>]*>/\n$&\n/ig;
 s/<\/?HEAD[^>]*>/\n$&\n/ig;
 s/<\/?BODY[^>]*>/\n$&\n/ig;
 s/<\/?PRE[^>]*>/\n$&\n/ig;
 s/<\/?TR[^>]*>/\n$&\n/ig;

 # Begin a line: <Hn>, <TITLE>, <LI>, <!-- comments -->.
 s/\<H[1-9][^>]*\>/\n$&/ig;
 s/<TITLE[^>]*>/\n$&/ig;
 s/<LI[^>]*>/\n$&/ig;
 s/\<!--[^>]*\>/\n$&/ig;

 # End a line: </Hn>, </TITLE>.
 s/<\/H[1-9][^>]*>/$&\n/ig;
 s/<\/TITLE[^>]*>/$&\n/ig;
```

```
  if (m/<PRE[^>]*>/i) { $in_pre_section = 1;}
  if (m/<\/PRE[^>]*>/i) { $in_pre_section = 0;}

  if ($in_pre_section == 1) {
   # We are in a <PRE> section; just print it out.
   print;
   print "\n";
  } else {
   # We aren't in a <PRE> section.
   # Previous line didn't have a newline, output a separator.
   s/^ *//;  # Squeeze out leading and trailing spaces.
   s/ *$//;
   if ($_ eq "") {$_ = "\n";}; # If nothing on this line, make it a newline.

   if ($previous_line_had_no_separator && (! m/^\n/)) { print ' '; }

   print;

   if (m/[\n \t]$/ ) { $previous_line_had_no_separator = 0;}
   else              { $previous_line_had_no_separator = 1;}
  }

}

print "\n\n";
```

LIST OF REFERENCES

[AFSAB 1994] Air Force Scientific Advisory Board. 1994 (1993 Summer Study). *Information Architectures that Enhance Operational Capability in Peacetime and Wartime*. Washington, DC: Air Force Printing Office.

[ASB 1995] Army Science Board. April 1995 (1994 Summer Study). *Technical Information Architecture for Command, Control, Communications and Intelligence*. Final Report. Washington, DC: Department of the Army, Assistant Secretary of the Army (Research, Development and Acquisition).

[BMDO 1995] Ballistic Missile Defense Organization (BMDO). May 25, 1995. *National Missile Defense Battle Management, Command, Control and Communications Information Architecture (NMD BM/C3 IA)*. Washington, DC: Ballistic Missile Defense Organization.

[Bush 1945] Bush, Vannevar. July 1945. "As We May Think." *Atlantic Monthly*. pp. 101-108.

[DSB 1993] Defense Science Board. December 1993 (1993 Summer Study). *Report of the DSB Task Force on Global Surveillance*. Washington, DC: OUSD(A&T). DTIC Accession Number AD-C051 840. Classified SECRET.

[Fife 1995] Fife, Dennis W. et al. February 1995. Memorandum for Richard Iliff, "Draft Review of BMDO Information Architecture Models." Alexandria, VA: Institute for Defense Analyses.

[Nelson 1981] Nelson, Ted. 1981. *Literary Machines*. Sausalito, CA: Mindful Press.

[Nielson 1990] Nielsen, Jakob. 1990. *Hypertext & Hypermedia*. San Diego, CA: Academic Press, Inc.

[Rumbaugh 1991] Rumbaugh, James, et al. 1991. *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice-Hall.

[Schwartz 1994] Schwartz, Randal L. 1994. *Learning Perl.* Sebastopol, CA: O'Reilly and Associates.

[US 1995] 10 United States Code Annotated, §2431, Section 233(b)(2). 1995. Also known as short title, "Missile Defense Act of 1991, as amended, section 233(b)(2)."

[Wall 1992] Wall, Larry and Randal L. Schwartz. 1992. *Programming Perl.* Sebastopol, CA: O'Reilly and Associates.

LIST OF ACRONYMS

BMD	Ballistic Missile Defense
BMDO	Ballistic Missile Defense Organization
BM/C3	Battle Management/Command, Control, and Communications
CASE	Computer-Aided Software (or System) Engineering
CERN	Conseil Européen pour la Recherche Nucléaire (European Laboratory for Particle Physics)
CONOPS	Concept of Operations
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
IA	Information Architecture
IDA	Institute for Defense Analyses
NCSA	National Center for Supercomputing Applications
NMD	National Missile Defense
OTS	Off-The-Shelf
PC	Personal Computer
Perl	Practical Extraction and Report Language
RTF	Rich Text Format
SEDD	System Engineering Design Data Base
URL	Uniform Resource Locator
WWW	World Wide Web
W3C	World Wide Web Consortium

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1995	Final

4. TITLE AND SUBTITLE
Translating System Development Documents to Hypertext: A Generic Process and Lessons Learned from a Prototype

5. FUNDING NUMBERS
DASW01-94-C-0054
Task Order T-R2-597.2

6. AUTHOR(S)
David A. Wheeler, Dennis W. Fife

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Defense Analyses (IDA)
1801 N. Beauregard St.
Alexandria, VA 22311-1772

8. PERFORMING ORGANIZATION REPORT NUMBER
IDA Paper P-3166

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
BMDO/DB
The Pentagon, Room 1E168
Washington, DC 20301-7100

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, unlimited distribution: March 27, 1996.

12b. DISTRIBUTION CODE
2A

13. ABSTRACT (Maximum 200 words)
This paper presents a generic process and lessons learned for developing hypertext versions of system development documents (e.g., requirements documents and design documents). System development documents describe many complex subcomponent interrelationships, making the information difficult to understand. What is desired is a tightly woven document, that is, a hypertext document with a large number of internal hypertext interconnections that help the reader examine subcomponents and their interrelationships on line. The generic process for developing tightly woven documents and the lessons learned described in this paper are based on our experiences in developing a prototype tightly woven document for the Ballistic Missile Defense Organization (BMDO). We recommend that BMDO consider, at document conception, developing and distributing key system development documents as tightly woven hypertext documents using the information discussed in this paper.

14. SUBJECT TERMS
Hypertext, HTML, Prototype, Information Architecture, Tightly Woven Document, System Development Documentation, BMD.

15. NUMBER OF PAGES
76

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102